International University of Sarajevo

WebShop Database

Introduction to Database Management

(Project I)

Professor: Dr. Ali Abd Almisreb

Students: Asja Bašović, Sara Avdić

Sarajevo, April 11, 2025

Table of Contents

1. Introduction 4
2. Software used 6
3. System Analysis 7
4. External level design 8
5. Conceptual level design 12
5.1. The conceptual and logical design 12
5.2. The physical design 16
5.3. Tables overviews 16
5.3.1 User Table Overview 16
5.3.2 Address Table Overview 17
5.3.3 Orders Table Overview 19
5.3.4 OrderItems Table Overview 21
5.3.5 Payments Table Overview 22
5.3.6 Products Table Overview 23
5.3.7 Sizes Table Overview 24
5.3.8 Colors Table Overview 25
5.3.9 Product Variants Table Overview 26
5.3.10 Categories Table Overview 27
6. Internal level design 29
7. Relational Algebra and Calculus 30
7.1 Relational Algebra 30
7.1.1 Selection 30
7.1.2 Projection 30
7.1.3 Cartesian Product 31
7.1.4 Union 31
7.1.5 Intersection 32
7.1.6 Set Difference 33
7.2 Relational Calculus 33
7.2.1 Selection 33
7.2.2 Projection 34

7.2.3 Cartesian Product 34
7.2.4 Union 34
7.2.5 Intersection 34
7.2.6 Set Difference 35
8. Data Normalization 36
9. User and System Roles 37
10. Security and Data Integrity 39
10.1 Password Hashing 39
10.2. Not Storing Credit Card Information 39
10.3 Protecting User Addresses 40
11. Procedures created and other SQL commands used 41
12. Testing and validation 43
13. Conclusion 44
14. Resources 45

1. Introduction

E-commerce, also known as electronic commerce, is the process of buying and selling goods and services, or the transmitting of funds or data, mainly over the internet (Hashemi-Pour, 2023). It has changed the way people shop, providing convenience and accessibility to a wide range of products. Managing an online shop involves handling large quantities of data, including customer information, product inventory, and transaction records. Without a well-structured database, businesses may face issues such as inconsistent data, difficulty in tracking stock levels, and inefficiencies in processing orders. A well-structured database is the backbone of any successful web shop, ensuring efficient management of users, orders, payments, and inventory. Hence, his project focuses on designing and implementing a relational database for an online clothing store. The store in question only sells clothing items for women that are ethically made. This means that the store has a relatively small but loyal pool of customers. A well-designed database is essential for tracking product availability, processing customer orders, managing payments, and ensuring seamless user interactions. This project addresses these challenges by creating a scalable and normalized database, reducing redundancy and ensuring efficient data retrieval. The system will support core functionalities such as user registration, product browsing, order placement, inventory tracking, and payment processing. The database will store essential details, including product variants (size, color), user addresses, and order statuses, ensuring smooth operations within the online store. This project aims to address the challenges of managing an online shop by developing a relational database that ensures:

- Efficient storage of user, product, and order information.
- Accurate inventory management with product variants.
- Seamless order tracking from placement to delivery.
- Data integrity and security, preventing redundant and inconsistent data.

The primary objectives of this project are:

1. Design a relational database that supports an online clothing store, ensuring efficient storage and retrieval of data.
2. Ensure normalization to eliminate redundancy and maintain data consistency.
3. Implement data integrity constraints to prevent invalid data entry.
4. Enable order processing and inventory tracking, ensuring each product variant (size and color) has an accurate stock level.
5. Support multiple user addresses, allowing customers to select different shipping locations.

6. Facilitate payment processing, tracking different payment methods and statuses.

Included in the Scope of the project

- Product catalog with detailed product variants (sizes, colors).
- Inventory management (stock tracking for product variants).
- Payment processing (recording payment methods and statuses).
- Address management (supporting multiple addresses per user).

This report will detail the conceptual, logical, and physical design of the database, explaining each entity, relationship, and constraint applied. The report also covers the testing stage and the overall process of building the database. The final implementation will ensure a scalable and efficient database system that can support a fully functional e-commerce platform.

2. Software used

The domain of this project is online retail for clothing. In today's digital era, without a proper database, companies may struggle with data inconsistency, inefficient inventory tracking, and order processing delays. Hence, it has been decided to use MySQL for this project, because it ensures data integrity and consistency through relational constraints (primary keys, foreign keys), it provides efficient querying and indexing, allowing quick retrieval of needed details, it supports transactions and ACID compliance, ensuring secure and reliable order and payment processing, and it allows scalability and integration, making it a practical choice for an e-commerce background. In addition, XAMPP and phpMyAdmin were used to create and host the database, and other software has also been used such as DBDiagram (https://dbdiagram.io/home) to make the diagrams in this report. Data used in the database is synthetic and generated using AI tools (primarily ChatGPT which is listed in the references). This method of data collection has been decided upon since it would be difficult to collect real data which is not needed to make the model functional.

3. System Analysis

To ensure the efficiency and functionality of the e-commerce database, the system must meet several key requirements. These can be divided into functional and non-functional requirements. The database must support the following functionalities:

- User Management: Allow users to register, log in, and manage their personal information, including multiple addresses.
- Product Catalog: Store and retrieve product details, including names, prices, categories, and variations (sizes, colors).
- Inventory Management: Maintain stock levels for each product variant, ensuring accurate inventory tracking.
- Order Processing: Allow users to place orders, track order statuses, and manage order history.
- Payment Handling: Record payment details, track payment statuses.
- Shipping and Address Management: Enable users to manage multiple addresses and select a shipping destination at checkout.

The non-functional requirements include:

- Scalability: The system must handle an increasing number of products, users, and transactions efficiently.
- Data Integrity: Ensure data consistency through constraints, normalization, and ACID compliance.
- Security: Protect user credentials, prevent unauthorized access, and ensure secure payment handling.
- Performance Optimization: Use indexing and efficient queries to optimize response times for product searches and order processing.
- Reliability: Ensure minimal downtime and data redundancy to maintain a smooth shopping experience.

Throughout the report, it will be explained how each decision and implementation contribute to one or more of the functionalities listed above.

4. External level design

The external level design of the database focuses on presenting specific, role-appropriate views of the data to various users while hiding the underlying complexity of the full schema. It defines how different users (which are explained in more detail in the paragraph *User and System Roles*) interact with the system based on their roles and access privileges. The main goal was to restrict access to the actual tables in the database and only allow specific roles access to the needed views. This has been done for security and simplicity reasons. In this database, five views are created:

1. OrderDetails

```
CREATE VIEW OrderDetailsView AS
  SELECT
    o.orderID,
    o.userID,
CONCAT(p.productName, ' (', s.sizeName, ', ', c.colorName, ')') AS productDescription,
oi.quantity,
p.price,
(oi.quantity * p.price) AS itemTotal
FROM orderitems oi
JOIN orders o ON oi.orderID = o.orderID
JOIN productvariants pv ON oi.productVariantID = pv.productVariantID
JOIN products p ON pv.productID = p.productID
JOIN sizes s ON pv.sizeID = s.sizeID
JOIN colors c ON pv.colorID = c.colorID
WHERE oi.isDeleted = FALSE;
```

2. MergeOrders

```
CREATE VIEW MergedOrdersView AS
SELECT
        o.orderID,
        GROUP_CONCAT(oi.quantity) AS quantities,
        GROUP_CONCAT(p.name) AS product_names,
        GROUP_CONCAT(s.sizeName) AS sizes,
        GROUP_CONCAT(pv.color) AS colors, -
```

```
        SUM(oi.quantity * p.price) AS total_price -
FROM
        orders o
JOIN
        orderitems oi ON o.orderID = oi.orderID
JOIN
        productvariant pv ON oi.productVariantID = pv.productVariantID
JOIN
        product p ON pv.productID = p.productID
JOIN
        size s ON pv.sizeID = s.sizeID
GROUP BY
        o.orderID;
```

3. UserAccountsView

```
CREATE VIEW UserAccountsView AS
SELECT
userID,
firstName,
lastName,
email
FROM Users
WHERE isDeleted = FALSE;
```

4.UserOrderHistoryView

```
CREATE VIEW UserOrderHistoryView AS
SELECT
o.orderID,
 o.userID,
 o.orderDate,
o.status AS orderStatus,
p.paymentID,
p.paymentMethod AS paymentMethod,
p.paymentStatus AS paymentStatus,
```

```
p.transactionDate,
a.city,
a.country
FROM orders o
LEFT JOIN payments p ON o.orderID = p.orderID
LEFT JOIN addresses a ON o.addressID = a.addressID
WHERE o.isDeleted = FALSE;
```

5. InventoryMonotoringView

```
CREATE VIEW InventoryMonitoringView AS
 SELECT
 pv.product_variant_id,
 pv.product_id,
 p.product_name,
 pv.variant_name,
 pv.stock_quantity
FROM
        product_variants pv
JOIN
        products p ON pv.product_id = p.product_id
WHERE
        pv.stock_quantity < 20;
```

- The Order Details View allows the product manager to see detailed financial information. It multiplies the price of each product by the quantity sold, providing clear insight into revenue. It also includes details such as product name, size, and color for accurate reporting.
- The Merge Orders View consolidates all items within a single order into one field, separated by commas, and displays the total cost per order. Other details such as size and color can also be seen. This view is useful for order processing staff or customer support teams to track orders at a glance.

- The User Accounts View shows details such as the user's first and last name and email . This view is accessible to customer support or administrators who manage user accounts.
- The User Order History View displays user's ID and all orders that were made, their payment and delivery status, transaction date and address of shipping.
- The Inventory Monitoring View highlights all product variants where the stock level is below 20. This view is crucial for inventory managers to ensure timely restocking and avoid product unavailability.

Access to these views is granted based on roles: administrators have full access to all views, product managers may access the inventory and earnings views, and customer service agents can access the order summary and user information views. This structure enhances security, supports role-based access control, and simplifies user interactions by only exposing the relevant parts of the database.

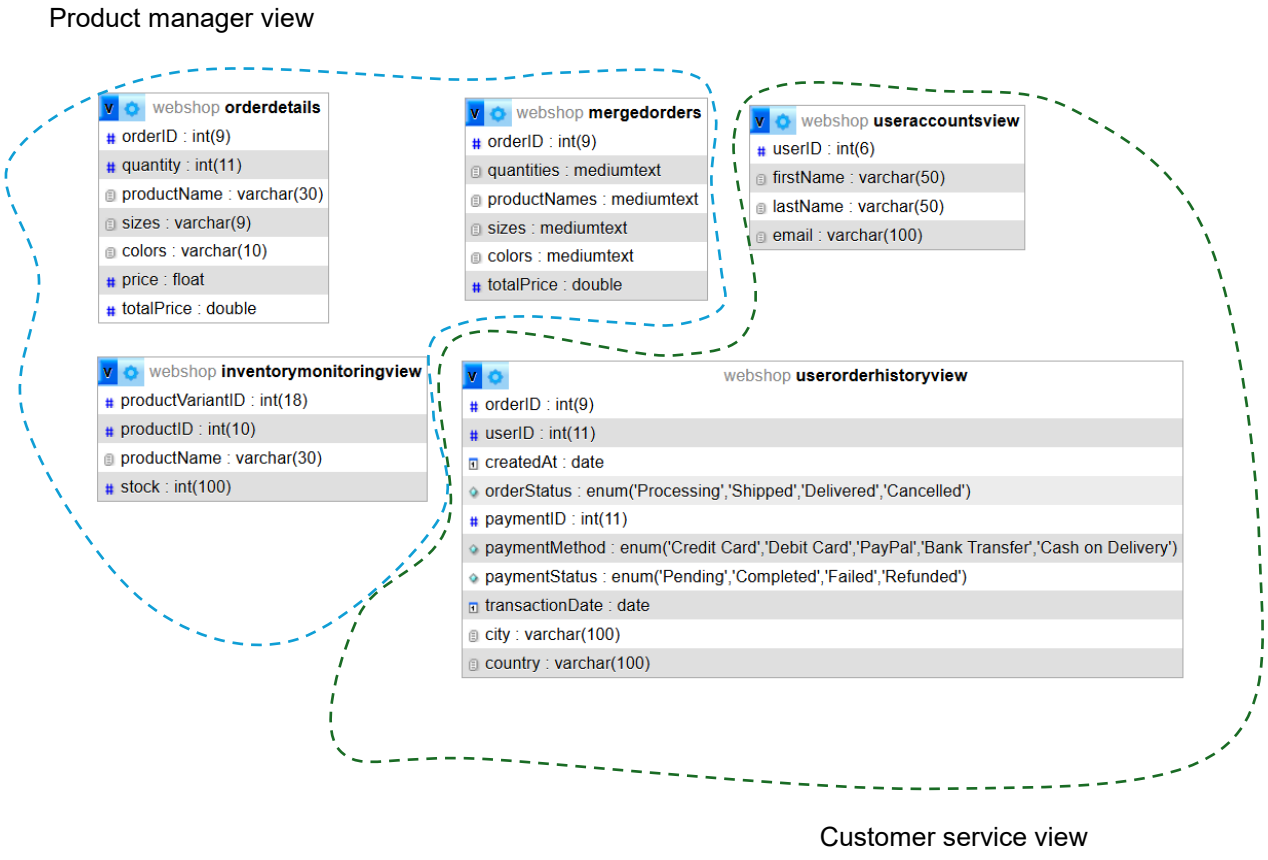

Figure 1, Views Diagram

5. Conceptual level design

5.1. The conceptual and logical design

The conceptual design defines the high-level structure of the database, outlining the key entities and their relationships. The Entity-Relationship (ER) Diagram represents the main components of the system and how they interact.

Below are the relations and their descriptions:

1. User: Represents registered users of the web shop. Each user can have multiple addresses.
2. Address: Stores shipping addresses associated with users. A user can have multiple addresses, but each address belongs to only one user.
3. Product: Represents items sold in the web shop. Products belong to a category and have multiple variations.
4. Category: Groups products into specific types (e.g., shirts, pants, accessories).
5. Size: Defines available size options for products (e.g., S, M, L, XL).
6. Color: Defines available color options for products.
7. ProductVariantStock: Represents product variants based on size and color, tracking their stock levels.
8. Order: Stores details of customer purchases. Each order belongs to one user and contains multiple order items.
9. OrderItem: Represents products within an order, linking orders to specific product variants.
10. Payment: Tracks payment details and statuses for orders. An order can have only one payment record.

Relationships that are present between the relations:

- One-to-Many Relationships:
    - A user can place multiple orders, but an order belongs to only one user.
    - A user can have multiple addresses, but each address is linked to only one user.
    - A category can contain multiple products, but each product belongs to only one category.
    - A product can have multiple product variants (sizes and colors), but each variant belongs to one product.
    - An order contains multiple order items, but each order item belongs to one order.
- Many-to-Many Relationships (Resolved Using Bridge Tables):

- Products can have multiple sizes and colors, but each combination is stored in the ProductVariantStock table.
- Orders contain multiple products, and each product can be part of multiple orders. This is managed through the OrderItem table.

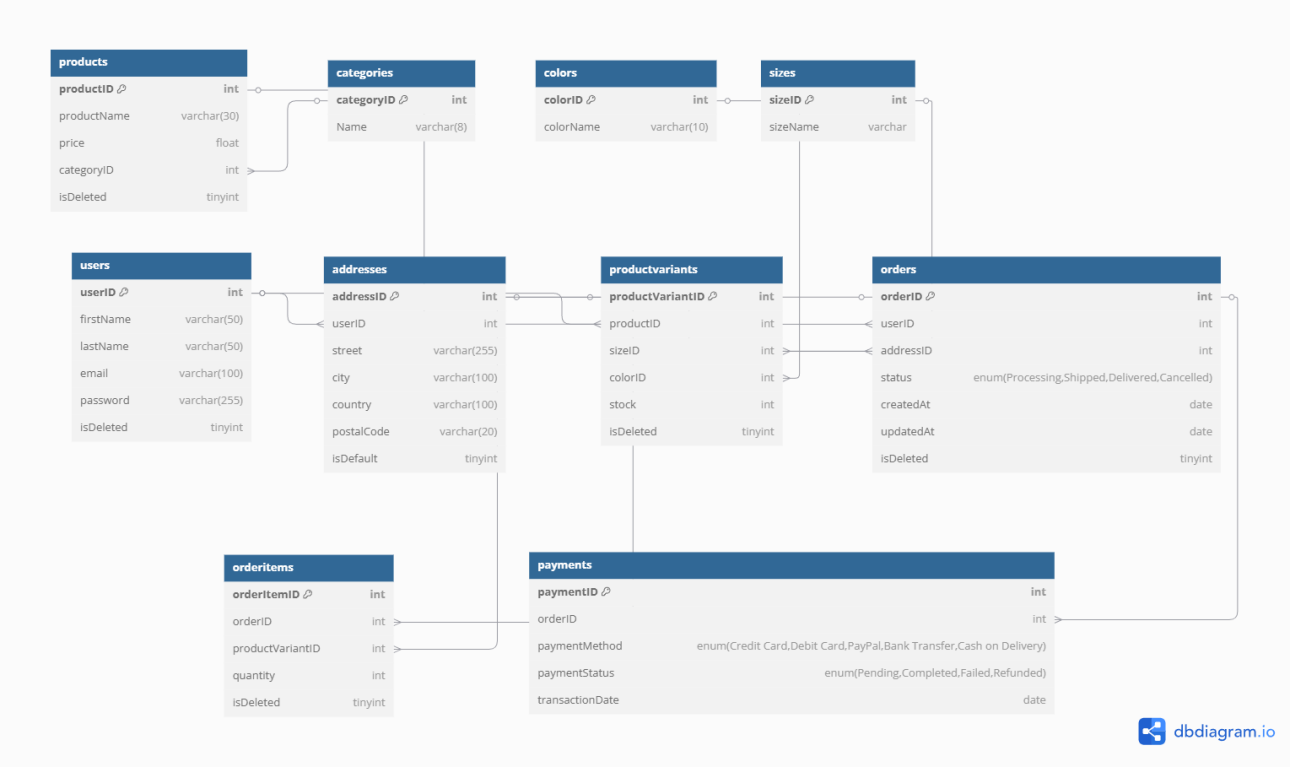

Figure 2, Logical Design

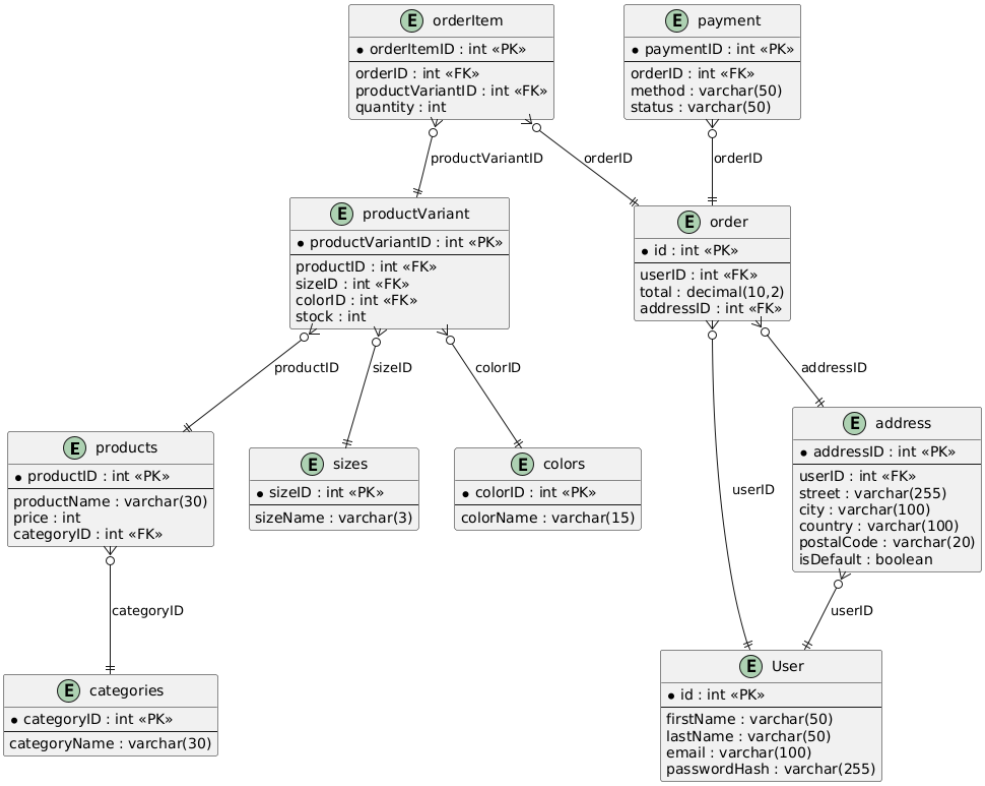

Figure 3, Entity - Relationship Diagram

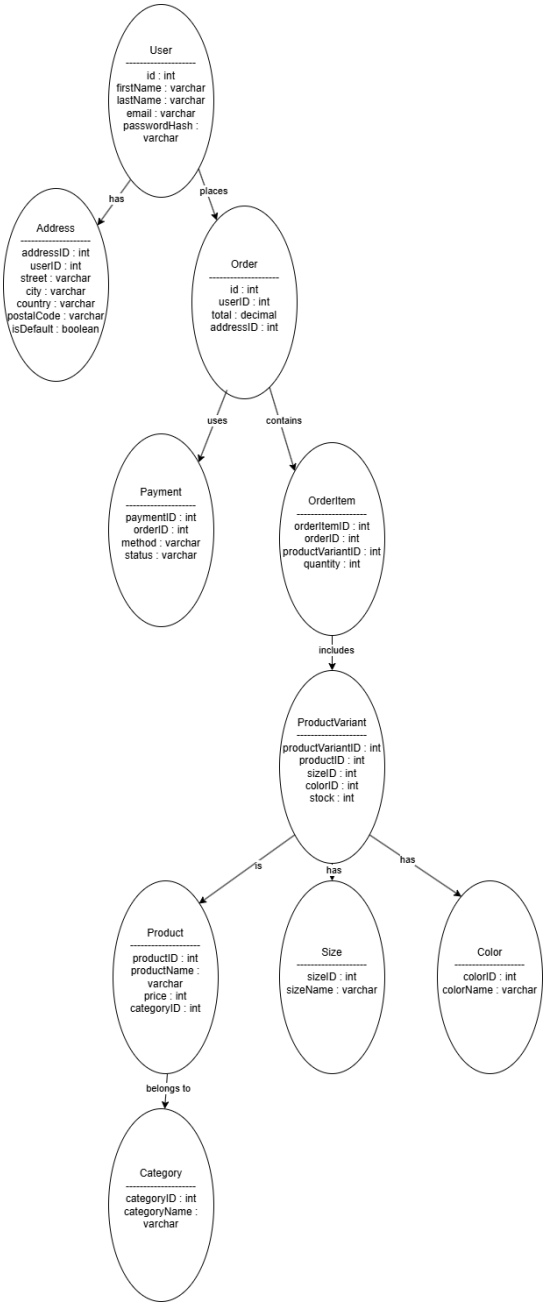

Figure 4, Semantic Net

5.2. The physical design

The physical design focuses on how the database will be implemented in MySQL and is hosted locally on a computer, including storage optimization, indexing strategies, and security measures. When it comes to the indexing strategy:

- Primary keys are indexed automatically.
- Foreign keys will be indexed for efficient joins and queries.
- Additional indexes may be added for frequently searched fields, such as productName in the Product table.

In addition, VARCHAR is used for text fields (e.g., names, emails, addresses) to optimize storage, DECIMAL(10,2) is used for price and total amounts to ensure precision in financial transactions, and ENUM is used for predefined statuses like order status and payment status. Moreover, the security measures implemented into the database include:

- Passwords are assumed to be using hashing (SHA-256 or bcrypt) at application level and are stored in that format.
- User authentication follows best practices to prevent SQL injection and data breaches.
- Role-based access control (RBAC) ensures database access is restricted based on user roles.

5.3. Tables overviews

Below the explanation for each table can be found:

5.3.1 User Table Overview

The User table is a core relation in the database and it stores information about the individuals who register on the web shop platform. The information stored is required for authentication and order processing. Each tuple in the User relation corresponds to a unique individual using the system. This uniqueness is enforced using the email attribute as each email can have only one registered user. The degree of the relation is 5, and the initial cardinality is 20 but it can increase/decrease as users register/delete accounts. The attributes stored are:

- userID (INT, PRIMARY KEY, AUTO_INCREMENT): A unique identifier for each user. It is automatically generated to ensure uniqueness and serves as the primary key.
- firstName (VARCHAR(50), NOT NULL): Stores the user's first name. The maximum length is 50 characters, and it cannot be left empty to ensure complete user records.

- lastName (VARCHAR(50), NOT NULL): Stores the user's last name with the same constraints as the first name.
- email (VARCHAR(100), UNIQUE, NOT NULL): Stores the user's email address, which serves as a unique identifier for login purposes. The UNIQUE constraint ensures that no two users can register with the same email.
- password (VARCHAR(255), NOT NULL): Stores the user's password in a hashed format for security. The field length is set to 255 to accommodate various hashing algorithms like bcrypt.
- isDeleted (tinyINT, NOT NULL): Stores either True(1) or False(0); checking if user's account is deleted.

The constraints and validation rules applied to the relation are:

- Primary Key: userID uniquely identifies each user.
- NOT NULL Constraints: Applied to all fields to ensure complete and valid user data.
- UNIQUE Constraint: Enforced on email to prevent duplicate accounts.

The domain restrictions applied to the relation are:

- firstName and lastName accept only alphabetic characters (additional validation can be applied at the application level). The domain of firstName and lastName is a set of valid names, typically alphabetic with optional spaces or hyphens.
- email must follow standard email formatting rules (checked using REGEX at the application level). The domain of email is a set of valid email addresses in the format username@domain.com.
- password must be hashed before storage to prevent unauthorized access. The domain of password consists of hashed strings that follow cryptographic security standards.

Relationships between the User relation and other relations are:

- One-to-Many Relationship with Orders: Each user can place multiple orders, establishing a 1:N (one-to-many) relationship. The userID serves as a foreign key in the Orders table.
- One-to-Many Relationship with Address: A user can have multiple saved addresses, making this another 1:N relationship, where userID is a foreign key in the Address table.

5.3.2 Address Table Overview

The Address table is a relation that stores location details associated with users, enabling them to save multiple shipping addresses for order deliveries. The relation ensures flexibility by allowing users to manage different addresses, such as home, work, or alternative shipping destinations. Each address is

linked to a user, establishing a one-to-many (1:N) relationship where a single user can have multiple addresses, but each address belongs to only one user. The degree of the relation is 7, and the initial cardinality is 22 but it can increase/decrease as users register addresses or delete accounts. The attributes stored are:

- addressID (INT, PRIMARY KEY, AUTO_INCREMENT): A unique identifier for each address, automatically incremented to ensure uniqueness.
- userID (INT, NOT NULL, FOREIGN KEY): References the User table, linking the address to a specific user. The NOT NULL constraint ensures that every address is associated with a user.
- street (VARCHAR(255), NOT NULL): Stores the street name and house number. The maximum length is set to 255 characters to accommodate detailed address entries.
- city (VARCHAR(100), NOT NULL): Stores the city name, with a length limit of 100 characters. Only alphabetic characters and spaces are expected.
- country (VARCHAR(100), NOT NULL): Stores the country name, with similar constraints as city.
- postalCode (VARCHAR(20), NOT NULL): Stores the postal code, allowing flexibility for different formats (e.g., numeric, alphanumeric, hyphenated).
- isDefault (BOOLEAN, DEFAULT FALSE, NOT NULL): A flag indicating whether this address is the user's default shipping address. A value of TRUE means this is the primary address used for orders unless the user selects another one.

The constraints and validation rules applied to the relation are:

- Primary Key: addressID uniquely identifies each address.
- Foreign Key Constraint: userID references the User table, ensuring that every address is associated with a valid user.
- NOT NULL Constraints: Applied to all fields to ensure completeness and prevent empty values.

The domain restrictions applied to the relation are:

- street, city, and country must contain valid location names and can include spaces, but not special characters like @ or #. The domain of city and country consists of valid place names, typically alphabetic with spaces.
- postalCode allows different country-specific formats, including numbers and letters. The domain of postalCode allows country-specific formatting (validated at the application level).
- isDefault is a Boolean (TRUE or FALSE), ensuring only valid values are stored. The domain of isDefault is strictly {TRUE, FALSE}.

Relationships between the Address relation and other relations are:

- One-to-Many Relationship with Users: Each user can have multiple addresses, but an address belongs to only one user (1:N relationship).
- One-to-Many Relationship with Orders: Each order must have a shipping address, creating a 1:N relationship where an address can be referenced in multiple orders.

5.3.3 Orders Table Overview

The Orders table is a relation that stores information about customer purchases in the web shop. Each order is associated with a specific user and a shipping address, and it includes details such as the total amount and order status. This table plays a crucial role in managing transactions, tracking order fulfilment, and maintaining customer purchase history. The Orders table establishes relationships with multiple other entities, including users, addresses, orderitems, and payments, ensuring a structured and normalized approach to order management. The degree of the relation is 7, and the initial cardinality is 18 but it can increase as users place more orders. The attributes stored are:

- orderID (INT, PRIMARY KEY, AUTO_INCREMENT): A unique identifier for each order. The AUTO_INCREMENT ensures sequential numbering of orders.
- userID (INT, NOT NULL, FOREIGN KEY): References the User table, indicating which user placed the order. This enforces a one-to-many (1:N) relationship, as a user can place multiple orders, but each order belongs to only one user.
- addressID (INT, NOT NULL, FOREIGN KEY): References the Address table, storing the shipping address for the order.
- total (DECIMAL (10,2), NOT NULL, CHECK (total >= 0)): Stores the total amount of the order. The DECIMAL (10,2) format ensures accurate representation of currency values, while the CHECK constraint prevents negative totals.
- status (ENUM('Processing', 'Shipped', 'Delivered', 'Cancelled') NOT NULL DEFAULT 'Processing'): Tracks the current state of the order. The ENUM data type restricts values to predefined statuses, ensuring data consistency.
- createdAt (DATETIME, NOT NULL DEFAULT CURRENT_TIMESTAMP): Stores the timestamp of when the order was placed. The default value ensures that every order has a recorded creation time.

- updatedAt (DATETIME, NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP): Records the last time the order status was updated. This automatically updates whenever a modification occurs.
- isDeleted (tinyINT, NOT NULL): Stores either True(1) or False(0); checking if order is deleted.

The constraints and validation rules applied to the relation are:

- Primary Key: orderID uniquely identifies each order.
- Foreign Keys:
    - userID references User(userID), ensuring that each order is associated with a valid user.
    - addressID references Address(addressID), enforcing that each order has a valid shipping address.
- NOT NULL Constraints: Prevents missing values in crucial fields such as userID, total, addressID, status, and timestamps.
- CHECK Constraint: Ensures that total is never negative.
- ENUM Constraint: Limits status to valid predefined values, preventing incorrect status entries.

The domain restrictions applied to the relation are:

- The domain of status is {Processing, Shipped, Delivered, Cancelled}, ensuring controlled order progression.
- The domain of total includes all non-negative decimal values, representing valid purchase amounts.
- The domain of createdAt and updatedAt consists of valid timestamps, ensuring accurate tracking of order history.

Relationships between the Address relation and other relations are:

- One-to-Many Relationship with users (1:N): A user can place multiple orders, but each order belongs to a single user.
- One-to-Many Relationship with addresses (1:N): An order must have a shipping address, but the same address can be used for multiple orders.
- One-to-Many Relationship with orderitems (1:N): Each order contains multiple items, stored in the orderitems table.
- One-to-One Relationship with payments (1:1): Each order has exactly one payment record, ensuring that payment details are tracked separately.

5.3.4 OrderItems Table Overview

The orderitems table is a relation that represents the products included in each order. Since an order can contain multiple products, and a single product can appear in multiple orders, this table handles the many-to-many (M:N) relationship between Orders and Product Variants and ensures that the database is normalised. Additionally, it ensures that every product purchased in an order is tracked individually, including its quantity and the specific variant (size and color). The degree of the relation is 4, and the initial cardinality is 30 but it can increase as users place more orders and order more items. The attributes stored are:

- orderItemID (INT, PRIMARY KEY, AUTO_INCREMENT): A unique identifier for each order item entry. The AUTO_INCREMENT ensures unique values.
- orderID (INT, NOT NULL, FOREIGN KEY): References the Orders table, indicating which order this item belongs to.
- productVariantID (INT, NOT NULL, FOREIGN KEY): References the ProductVariantStock table, linking to a specific size and color of a product.
- quantity (INT, NOT NULL, CHECK (quantity > 0)): Represents the number of units of this product variant in the order. The CHECK constraint ensures that only positive values are allowed.
- isDeleted(tinyINT, NOT NULL): Stores either True(1) or False(0); checking if order item is deleted.

The constraints and validation rules applied to the relation are:

- Primary Key: orderItemID uniquely identifies each row in the table.
- Foreign Keys:
    - orderID references Orders(orderID), ensuring that each item belongs to a valid order.
    - productVariantID references productvariants (productVariantID), ensuring that the product variant exists.
- NOT NULL Constraints: Ensures that every item entry has a valid order reference, product variant, and quantity.
- CHECK Constraint: Prevents negative or zero quantities, ensuring valid order entries.

The domain restrictions applied to the relation are:

- The domain of quantity consists of positive integers {1, 2, 3, ...}, ensuring only valid purchase amounts.
- The domain of orderID ensures that each order item is linked to a valid order.

- The domain of productVariantID ensures that each order item corresponds to an existing product variant.

Relationships between the Address relation and other relations are:

- Many-to-One Relationship with Orders (M:1): Each order can have multiple order items, but each order item belongs to only one order.
- Many-to-One Relationship with Product Variants (M:1): Each order item is linked to a specific product variant (size and color), but the same variant can appear in multiple orders.

5.3.5 Payments Table Overview

The Payments table stores transaction details related to customer orders. Since each order must be paid for before it is processed, this table ensures that payments are properly recorded and linked to specific orders. It captures the payment method, status, and other necessary details to maintain transaction integrity. The relationship between Orders and Payments is one-to-one (1:1), as each order has exactly one corresponding payment. The degree of the relation is 7, and the initial cardinality is 18 but it can increase as users place more orders. The attributes stored are:

Attributes and Data Storage

- paymentID (INT, PRIMARY KEY, AUTO_INCREMENT): A unique identifier for each payment transaction. The AUTO_INCREMENT ensures that every payment entry has a unique ID.
- orderID (INT, NOT NULL, UNIQUE, FOREIGN KEY): References the Orders table, ensuring that each order is linked to exactly one payment. The UNIQUE constraint enforces a one-to-one relationship.
- method (ENUM('Credit Card', 'Debit Card', 'PayPal', 'Bank Transfer', 'Cash on Delivery') NOT NULL): Defines the payment method used by the customer. The ENUM constraint ensures only valid methods are stored.
- status (ENUM('Pending', 'Completed', 'Failed', 'Refunded') NOT NULL DEFAULT 'Pending'): Tracks the current state of the payment. The default value is 'Pending' until the transaction is successfully processed.
- transactionDate (DATETIME, NOT NULL DEFAULT CURRENT_TIMESTAMP): Records the exact time the payment was made. The default value ensures that every transaction is timestamped.

Constraints and Validation Rules

- Primary Key: paymentID uniquely identifies each payment entry.
- Foreign Key:
- orderID references orders (orderID), ensuring that each payment is associated with a valid order.
- UNIQUE Constraint: Ensures that an order cannot have multiple payments.
- NOT NULL Constraints: Ensures that no field contains missing values.
- ENUM Constraints:
- The method column restricts values to predefined payment options, preventing invalid payment types.
- The status column ensures payments follow a controlled status workflow (Pending → Completed / Failed / Refunded).

Cardinality and Relationships

- One-to-One Relationship with Orders (1:1): Each order has exactly one payment, and each payment corresponds to a single order.
- One-to-Many Relationship with Users (1:N) (Indirectly Through Orders): Since a user can place multiple orders, and each order has a payment, a user can have multiple payments recorded in the system.

Data Domain and Integrity

- The domain of method consists of {Credit Card, Debit Card, PayPal, Bank Transfer, Cash on Delivery}, ensuring only valid payment types.
- The domain of status consists of {Pending, Completed, Failed, Refunded}, ensuring payments follow a proper lifecycle.
- The domain of transactionDate ensures all payment records have a valid timestamp.

5.3.6 Products Table Overview

Products Table stores products available in the web shop. Each product has unique ID and name. Also, with every product there is price associated. One important note is that each product belongs to one category (in this case 9 categories) and is associated with categoryID in a relationship one to many. The degree of the relation is 4, and the initial cardinality is 45 but it can increase/decrease as admin adds/deletes products.

Attributes and Data Storage

- productID (INT, PRIMARY KEY, AUTO_INCREMENT): A unique identifier for each product. It is automatically generated to ensure uniqueness and serves as the primary key.

- productName (VARCHAR(30), NOT NULL): Stores the product's name. The maximum length is 30 characters, and it cannot be left empty.
- price (INT, NOT NULL): Stores price for each product. It cannot be left empty.
- categoryID (INT, INT, NOT NULL, FOREIGN KEY): Makes sure each product belongs to some category in Categories Table.
- isDeleted(tinyINT, NOT NULL): Stores either True(1) or False(0); checking if product is deleted.

Constraints and Validation Rules

- Primary Key: productID uniquely identifies each product.
- NOT NULL Constraints: Applied to all fields to ensure complete and valid data.

Domain Restrictions:

- productName accepts only alphabetic characters (additional validation can be applied at the application level).
- price can only be numeric value and cannot be negative.

Cardinality and Relationships

- One-to-Many Relationship with Orders: Each category can be placed for many products, establishing a 1:N (one-to-many) relationship. The categoryID serves as a foreign key in the Products Table.

Data Domain and Integrity

- The domain of productName is a set of valid names, typically alphabetic with optional spaces or hyphens.

5.3.7 Sizes Table Overview

Sizes Table stores sizes available in the web shop. Each size has unique ID and name. The degree of the relation is 2, and the initial cardinality is 3 but it can increase/decrease as admin adds/deletes sizes.

Attributes and Data Storage

- sizeID (INT, PRIMARY KEY, AUTO_INCREMENT): A unique identifier for each size. It is automatically generated to ensure uniqueness and serves as the primary key.
- sizeName (VARCHAR(3), NOT NULL): Stores the size name. The maximum length is 3 characters, and it cannot be left empty.

Constraints and Validation Rules

- Primary Key: sizeID uniquely identifies each size.
- NOT NULL Constraints: Applied to all fields to ensure complete and valid data.

Domain Restrictions:

- sizeName accepts only alphabetic characters (uppercase letters).

Data Domain and Integrity

- The domain of sizeName is a set of valid characters, typically alphabetic with optional spaces or hyphens.

5.3.8 Colors Table Overview

Colors Table stores colors available for each product in the web shop. Each color has unique ID and name. The degree of the relation is 2, and the initial cardinality is 11 but it can increase/decrease as admin adds/deletes colors.

Attributes and Data Storage

- colorID (INT, PRIMARY KEY, AUTO_INCREMENT): A unique identifier for each color. It is automatically generated to ensure uniqueness and serves as the primary key.
- colorName (VARCHAR(15), NOT NULL): Stores the color name. The maximum length is 15 characters, and it cannot be left empty.

Constraints and Validation Rules

- Primary Key: colorID uniquely identifies each size.
- NOT NULL Constraints: Applied to all fields to ensure complete and valid data.

Domain Restrictions:

- colorName accepts only alphabetic characters.

Data Domain and Integrity

- The domain of colorName is a set of valid characters, typically alphabetic with optional spaces or hyphens.

5.3.9 Product Variants Table Overview

The Product Variants table stores variants of each product in terms of sizes and colors. This table ensures that each product variant has unique ID (each size and each color is one product variant). Table stores productID, colorID and sizeID, all being foreign keys from products, colors and sizes tables.

Attributes and Data Storage

- productVariantID (INT, PRIMARY KEY, AUTO_INCREMENT): A unique identifier for each address, automatically incremented to ensure uniqueness.
- productID (INT, NOT NULL, FOREIGN KEY): References the User table, linking the address to a specific user. The NOT NULL constraint ensures that every address is associated with a user.
- sizeID (INT, NOT NULL, FOREIGN KEY): References the Sizes table, linking the size to a specific product. The NOT NULL constraint ensures that every product has a size.
- colorID (INT, NOT NULL, FOREIGN KEY): References the Colors table, linking the color to a specific product. The NOT NULL constraint ensures that every product has a color.
- stock (INT, NOT NULL): States stock quantity of each product and decreases as orders come in.
- isDeleted (tinyINT, NOT NULL): Stores either True(1) or False(0); checking if product variant is deleted.

Constraints and Validation Rules

- Primary Key: productVariantID uniquely identifies each address.
- Foreign Key Constraint: productID references the products table, sizeID references the Sizes table and colorID references the colors table, ensuring every product has size and color.
- NOT NULL Constraints: Applied to all fields to ensure completeness and prevent empty values.

Domain Restrictions:

- Stock must be an integer and is decreased as users place orders.

Cardinality and Relationships

- One-to-Many Relationship with products: Each product can be associated with many product variants, making it 1:N relationship.
- One-to-Many Relationship with sizes: Each size can be associated with many product variants, making it 1:N relationship.
- One-to-Many Relationship with colors: Each color can be associated with many product variants, making it 1:N relationship.

Data Domain and Integrity

- The domain of stock is strictly a positive integer.

5.3.10 Categories Table Overview

The Categories table stores categories of products available in the web shop. Table stores

Attributes and Data Storage

- productVariantID (INT, PRIMARY KEY, AUTO_INCREMENT): A unique identifier for each address, automatically incremented to ensure uniqueness.
- productID (INT, NOT NULL, FOREIGN KEY): References the User table, linking the address to a specific user. The NOT NULL constraint ensures that every address is associated with a user.
- sizeID (INT, NOT NULL, FOREIGN KEY): References the Sizes table, linking the size to a specific product. The NOT NULL constraint ensures that every product has a size.
- colorID (INT, NOT NULL, FOREIGN KEY): References the Colors table, linking the color to a specific product. The NOT NULL constraint ensures that every product has a color.
- stock (INT, NOT NULL): States stock quantity of each product and decreases as orders come in.

Constraints and Validation Rules

- Primary Key: productVariantID uniquely identifies each address.
- Foreign Key Constraint: productID references the Products table, sizeID references the Sizes table and colorID references the Colors table, ensuring every product has size and color.
- NOT NULL Constraints: Applied to all fields to ensure completeness and prevent empty values.

Domain Restrictions:

- Stock must be an integer and is decreased as users place orders.

Cardinality and Relationships

- One-to-Many Relationship with products: Each product can be associated with many product variants, making it 1:N relationship.
- One-to-Many Relationship with sizes: Each size can be associated with many product variants, making it 1:N relationship.
- One-to-Many Relationship with colors: Each color can be associated with many product variants, making it 1:N relationship.

Data Domain and Integrity

- The domain of stock is strictly a positive integer.

6. Internal level design

The internal level of the database focuses on the physical storage, data organization, indexing, and query optimization to ensure efficient performance and security. Data is stored using MySQL's structured storage methods. Indexes are applied to frequently queried columns, such as product names and order IDs, to improve search efficiency. Query execution plans help optimize SQL statements, ensuring minimal processing time for complex joins and lookups. Transaction management follows ACID principles, maintaining data consistency by ensuring that multi-step operations, such as order placements and payments, either fully execute or completely roll back in case of failure. To further optimize performance, table partitioning by date and indexing foreign keys are used in large tables like orders and orderitems. Security measures include encryption of sensitive data and access control through user permissions, ensuring that only authorized system components can access or modify data. These internal-level optimizations collectively enhance database efficiency, reliability, and security, making the web shop scalable and responsive for users.

7. Relational Algebra and Calculus

7.1 Relational Algebra

7.1.1 Selection

Get all products that cost more than 100.

$\sigma_{price>100}(products)$

```
SELECT * FROM products
WHERE price > 100;
```

	productID	productName	price	categoryID	isDeleted
Edit Copy Delete	17	Puffer Jacket	110	4	0
Edit Copy Delete	18	Coat	120	4	0
Edit Copy Delete	20	Fur Jacket	130	4	0

7.1.2 Projection

Get only the product names and prices.

$\pi_{productName,price}(products)$

```
SELECT productName, price FROM products;
```

productName	price
Skinny Jeans	40.25
Wide Leg Jeans	50
Mom-fit Jeans	45
Straight Leg Jeans	60
Boot Cut Jeans	50
Straight Leg Pants	60
High Waist Pants	65
Low Waist Pants	50
Wide Leg Pants	50
Plaid pants	70
Long Sleeve T-Shirt	30
Short Sleeve T-Shirt	20
V-cut T-Shirt	25
Sleeveless T-Shirt	20
Graphic T-Shirt	35
Denim Jacket	70
Puffer Jacket	110
Coat	120
Leather Jacket	80
Fur Jacket	130
Maxi Skirt	30
Midi Skirt	25
Mini Skirt	20
Pencil Skirt	35
Denim Skirt	40

7.1.3 Cartesian Product

Get all combinations of colors and sizes.

colors×sizes

```
SELECT * FROM colors
CROSS JOIN sizes;
```

colorID	colorName	sizeID	sizeName
1	White	1	S
1	White	2	M
1	White	3	L
2	Black	1	S
2	Black	2	M
2	Black	3	L
3	Red	1	S
3	Red	2	M
3	Red	3	L
4	Orange	1	S
4	Orange	2	M
4	Orange	3	L
5	Yellow	1	S
5	Yellow	2	M
5	Yellow	3	L
6	Green	1	S
6	Green	2	M
6	Green	3	L
7	Blue	1	S
7	Blue	2	M
7	Blue	3	L
8	Purple	1	S
8	Purple	2	M
8	Purple	3	L
9	Pink	1	S

7.1.4 Union

Get all countries where users have addresses or orders were shipped (via address).

$\pi_{country(}$address)$\cup\pi_{country}$(address)

```
SELECT country FROM address
UNION
SELECT country FROM address;
```

country
Vanuatu
France
Kuwait
United States Minor Outlying Islands
Swaziland
Faroe Islands
Denmark
Zimbabwe
Turkey
French Southern Territories
New Caledonia
Isle of Man
Cocos (Keeling) Islands
Togo
Niue
Myanmar
Colombia
Maldives
Greenland
Reunion
Czech Republic
Bulgaria
Turks and Caicos Islands
Niger
Central African Republic

7.1.5 Intersection

Get countries that are shared by user 1 and 2.

$\pi_{country}(\sigma_{userID=1}(address)) \cap \pi_{country}(\sigma_{userID=2}(address))$

SELECT country FROM address WHERE userID = 1

INTERSECT

SELECT country FROM address WHERE userID = 2;

country
Denmark
Reunion
South Africa
Zimbabwe

7.1.6 Set Difference

Get users who haven't placed an order.

$\pi_{userID,firstName,lastName}(\sigma_{userID \notin} / \pi_{userID}(orders)(users))$

```
SELECT userID, firstName, lastName FROM users
WHERE userID NOT IN (
SELECT userID FROM orders
);
```

	userID	firstName	lastName
Edit Copy Delete	3	Christopher	Johnson
Edit Copy Delete	4	William	Diaz
Edit Copy Delete	5	Toni	Daugherty
Edit Copy Delete	6	Michael	Russell
Edit Copy Delete	7	William	Montgomery
Edit Copy Delete	13	Michael	Turner
Edit Copy Delete	15	Gerald	Campos
Edit Copy Delete	17	Eric	Johnson
Edit Copy Delete	37	Christopher	Griffin
Edit Copy Delete	40	Maria	Nguyen
Edit Copy Delete	41	Ann	Green
Edit Copy Delete	43	Robin	Garcia
Edit Copy Delete	46	Kevin	Perez
Edit Copy Delete	47	Cynthia	Hernandez
Edit Copy Delete	48	Haley	Page
Edit Copy Delete	50	Melissa	Gonzales

7.2 Relational Calculus

7.2.1 Selection

Get all products with a price > 100.

TRC (Tuple Relational Calculus):

$\{ p \mid p \in products \land p.price > 100 \}$

DRC (Domain Relational Calculus):

$\{ <id, name, price, cid> \mid products(id, name, price, cid) \land price > 100 \}$

7.2.2 Projection

Get only the names of products.

TRC (Tuple Relational Calculus):

{ p.productName | p ∈ products }

DRC (Domain Relational Calculus):

{ <name> | ∃id ∃price ∃cid (products(productID, productName, price, cid)) }

7.2.3 Cartesian Product

Get all pairs of product and category.

TRC (Tuple Relational Calculus):

{ <p, c> | p ∈ products ∧ c ∈ categories }

DRC (Domain Relational Calculus):

{ <pid, pname, price, cid, catid, catname> |

products(pid, productName, price, cid) ∧ categories(catid, catname) }

7.2.4 Union

Get all user first names and all category names (into one set).

TRC (Tuple Relational Calculus), DRC (Domain Relational Calculus):

{ <x> | ∃id ∃lastName ∃email ∃ph ∃a (User(id, x, lastName, email, ph)) }

∪

{ <x> | ∃cid (categories(cid, x)) }

7.2.5 Intersection

Find category names that also appear as first names in users.

TRC (Tuple Relational Calculus), DRC (Domain Relational Calculus):

{ <x> | ∃cid (categories(cid, x)) ∧ ∃id ∃lastName ∃email ∃ph ∃a (users(userID, x, lastName, email, ph)) }

7.2.6 Set Difference

Find users who have not placed any orders.

TRC (Tuple Relational Calculus):

{ u | u ∈ User ∧ ¬∃o (o ∈ order ∧ o.userID = u.id) }

DRC (Domain Relational Calculus):

{ <id, firstName, lastName, email, ph> |

users(id, firstName, lastName, email, ph) ∧

¬∃oid ∃total (orders(oid, orderID, total)) }

8. Data 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.” (Microsoft Learn, 2024) Our database follows the principles of First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF) to achieve an optimal design. Application of 1NF in Our Database:

- In the User table, attributes like firstName, lastName, and email store single, indivisible values.
- The orderitems table ensures each row represents a single product within an order (no lists of products).
- The productvariants table ensures that each product variant has its own row instead of a list of sizes and colors in the Product table.

Application of 2NF in Our Database:

- In the orderitems table, productID depends entirely on orderID. If we stored productName here, it would violate 2NF because productName depends only on productID. Instead, we store product details separately in the Product table.
- The productvariants table ensures that sizeID and colorID are properly linked to specific products rather than being embedded in the Product table.

Application of 3NF in Our Database:

- The orders table stores only userID and addressID, rather than embedding full user details.
- The payments table stores orderID rather than duplicating payment details in the OrderTable.
- The products table links to categories using categoryID, rather than storing the category name in the Product table.

9. User and System Roles

The system will have two types of direct users: managers and administrators, each with distinct roles and privileges within the web shop. Additionally, although customers do not connect directly to the database, they are end-users of the system whose interactions with the application result in database transactions. They can visit the web shop to browse and purchase products. Their primary interactions with the system include registering an account, logging in, updating personal information, viewing available products, adding products to the cart, modifying cart items, and proceeding to checkout, etc. Customers cannot modify product details, inventory, or other administrative settings. The customer actions are outside of the scope of this project. The focus is on the direct users. Managers manage the operational aspects of the system to ensure smooth business operations. This project includes the roles of two types of managers:

- Product manager:
    - Product Management: Adding, updating, and deleting products, categories, sizes, and colors. Can make decisions based on the profit an item makes.
    - Viewing the total sales and making decisions about the products (ex.: deleting a product if sales are low).
    - Inventory Management: updating stock levels in the productvariants table through a view.
    - They are allowed to insert, delete (only by using the predefined method), select, and update product information.
    - Can only access the following views: orderDetails, mergeOrders, inventoryMonitoringView.
- Customer support manager:
    - Customer support-interact with the database indirectly through application-level tools that use specially designed views (userAccountsView, userOrderHistoryView).
    - These views allow them to access user profiles, order histories, and shipping information in a secure and structured way.
    - Their role does not require modification of data, so they are only granted read and select access to views that help resolve customer queries while protecting sensitive or internal information.

Can only access the following views: The above-mentioned administrators do not have full access to database operations and cannot place personal orders like customers. In this case, only the database administrator (Sara Avdic and Asja Basovic) has full access to the database. The database administrator has full access to the database for maintenance, optimization, and backup tasks. However, since we are

the ones that designed and implemented this database fully, it can also be said that we are designers and data administrators as well. Below is the sample code used to give appropriate permissions to a user (in this example, the user is the customer support manager):

```
GRANT SELECT ON UserAccountsView TO 'support_user'@'localhost';

GRANT SELECT ON UserOrderHistoryView TO 'support_user'@'localhost';

GRANT SELECT ON OrderDetailsView TO 'support_user'@'localhost';
```

10. Security and Data Integrity

Security and data integrity are crucial aspects of any database system, especially for a web shop that handles user data, order transactions, and payments. This section discusses the mechanisms implemented to ensure data validity, prevent security vulnerabilities, and protect sensitive user information. To maintain data integrity, various constraints are applied at the database level to prevent invalid or inconsistent data from being stored. The following constraints are implemented:

- NOT NULL Constraints - Ensures that critical fields are never left empty, preventing incomplete data from being stored. Example: The email field in the User table must always be filled.
- UNIQUE Constraints – Prevents duplicate values in columns that require uniqueness. Example: Each user must have a unique email.
- CHECK Constraints - Ensures that only valid values are inserted into a column. Example: Order status should only be one of the predefined values ('Processing', 'Shipped', 'Delivered').
- FOREIGN KEY Constraints - Maintains referential integrity by ensuring that referenced data exists. Example: The Order.userID must correspond to a valid User.id.

These constraints eliminate common data errors such as missing values, duplicate records, and invalid entries, ensuring that the database remains consistent and reliable. Moreover, handling sensitive data securely is a fundamental requirement for protecting user privacy and complying with data protection regulations. The following measures are implemented:

10.1 Password Hashing

User passwords are never stored in plain text.Instead, they are hashed using a strong algorithm like SHA-256 or bcrypt before being stored. Example: Hashing a password before inserting into the database:

```
INSERT INTO User (firstName, lastName, email, password)
VALUES ('John', 'Doe', 'john@example.com', SHA2('password123', 256));
```

Even if the database is compromised, attackers cannot easily recover the original password. This is done on application level.

10.2. Not Storing Credit Card Information

The database does NOT store credit card numbers for security reasons. Instead, payments are processed using third-party payment gateways (e.g., Stripe, PayPal).

10.3 Protecting User Addresses

User addresses are stored in a separate Address table, linked to user accounts via foreign keys. The Address table contains an isDefault column to allow users to mark a preferred address.Only authorized users (i.e., the customer or administrators) can access address information.Example of secure access control:

SELECT * FROM Address WHERE userID = ?;

This ensures that users can only access their own saved addresses.

11. Procedures created and other SQL commands used

This SQL block is a stored procedure or trigger (depending on the context), meant to safely soft-delete a product variant from an e-commerce database. In this context, there is a “isDeleted” attribute in the product and productvariant tables. If the product manager wishes to delete a certain product, they will only be able to soft delete it (mark it True in isDeleted) since the goal is not to lose data (such as previous orders) by completely removing a product from the database.

```
BEGIN DECLARE productID_var INT;

-- Check if the product variant is linked to any order with status 'Processing'
IF EXISTS (
   SELECT 1
   FROM orderitems oi
   JOIN orders o ON oi.orderID = o.orderID
   WHERE oi.productVariantID = input_productVariantID
   AND o.status = 'Processing'
) THEN
   -- Raise an error if the product variant is in a processing order
   SIGNAL SQLSTATE '45000'
   SET MESSAGE_TEXT = 'Cannot delete: Product variant is linked to a processing order';
ELSE
   -- Get the productID of this variant
   SELECT productID INTO productID_var FROM productvariants WHERE productVariantID =
input_productVariantID;


   -- Soft delete the product variant
   UPDATE productvariants
   SET isDeleted = TRUE, stock = 0
   WHERE productVariantID = input_productVariantID;


   -- Check if all variants of this product are now deleted
   IF NOT EXISTS (
      SELECT 1 FROM productvariants
      WHERE productID = productID_var AND isDeleted = FALSE
```

```
    ) THEN
        -- If no active variants remain, soft delete the product
        UPDATE products
        SET isDeleted = TRUE
        WHERE productID = productID_var;
    END IF;
END IF;


END
```

12. Testing and validation

Testing played a crucial role in verifying that the database system met all its functional, performance, and security requirements. The process began by conducting unit testing on all individual database components, including stored procedures, and views, as well as testing how data was stored in the tables. For example, we tested the SoftDeleteProductVariant procedure by attempting to delete product variants linked to orders with a "Processing" status, ensuring that it correctly prevented the deletion and raised an error, while allowing deletion for fulfilled or cancelled orders. This example usage can be seen in the report where the procedure is explained. We also tested a custom procedure that checks whether all variants of a product are soft-deleted, and if so, automatically marks the main product as deleted—this procedure was validated by inserting a variety of test data combinations. To assess the data validation rules, we tried inserting invalid entries such as null values in required fields (e.g., email in the users table), text values in numeric fields (e.g., quantity in orderitems), and duplicate entries in unique columns. The database appropriately rejected each of these actions, confirming the effectiveness of the constraints like NOT NULL, UNIQUE, and CHECK. We also validated referential integrity by attempting to delete parent records that were still referenced by child tables, which correctly resulted in foreign key constraint errors. To test the views, we decided to have two other people log in as customer support manager and product manager. They do not know the underlying structure of the database and can only access the appropriate views and procedures. We believed this was the best way to test the roles as it is how the database would be used. The customer service manager was given access to views related to users and orders, such as a view showing user details along with their latest order status and payment status. This role was tested by performing tasks such as retrieving customer orders, confirming if an order was paid, and reviewing contact details for communication. The product manager, on the other hand, was granted access to product and inventory-related tables and views. They used the InventoryStatus view to identify products with low stock (less than 20 units) and tested procedures that updated stock levels and marked items as deleted when necessary. We confirmed that both roles were able to perform the operations relevant to them while being restricted from accessing sensitive or unrelated data. Overall, through a combination of manual test scenarios, and real-user simulations, we ensured the robustness, accuracy, and security of the database system. The testing process not only confirmed that the system adhered to the design requirements but also helped identify and resolve minor logic issues early, improving the final quality of the application.

13. Conclusion

In conclusion, this project successfully designed and implemented a relational database system tailored for an online clothing store. The primary goal was to create a structured, efficient, and scalable database that supports core business operations such as product management, inventory tracking, and customer information management. Through the application of Entity-Relationship modeling, logical and physical database design, and normalization techniques, we ensured the integrity, consistency, and optimization of the data stored within the system. The system was carefully tested through a series of unit, validation, and user acceptance tests, with real-world scenarios simulated by assigning different roles to users such as product managers and customer service staff. This not only confirmed the functionality of features like soft deletion, dynamic views, and role-specific access but also demonstrated the practicality and usability of the system in a business environment. The use of views enabled abstraction at the external level, simplifying data access for specific users while preserving security and consistency. Furthermore, we implemented best practices in security, including constraints, limited privileges, and the concept of soft deletion to maintain historical data integrity. The database is designed to be extensible and can support integration with front-end applications or APIs, making it suitable for real-world deployment. Overall, the result is a robust, secure, and user-friendly system that enhances both administrative efficiency and customer service.

14. Resources

Hashemi-Pour, C. (2023). *e-commerce*. TechTarget. https://www.techtarget.com/searchcio/definition/e-commerce

Microsoft Learn. (2024). *Database normalization description - Microsoft 365 Apps*. Microsoft.com. https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description

dbdiagram.io - Database Relationship Diagrams Design Tool. (2025). Dbdiagram.io. https://dbdiagram.io/home

ChatGPT. (2025). Chatgpt.com. https://chatgpt.com/

DiagramDraw. (2025). Diagram.net https://diagram.net/