Take Care of My Clothes

1. Abstract

The rise of fast fashion has led to consumers engaging in excessive clothing consumption. Since clothing causes environmental pollution during the processes of production, consumption, and disposal, a solution to overcome this situation is necessary. This project identified the fundamental cause of these issues as the lack of efficient clothing management, rather than merely the rise of fast fashion. Therefore, to reduce excessive clothing consumption and prevent environmental pollution, a wardrobe database was developed. The database was designed based on the ER model and constructed using seven tables that satisfy 3NF. Interactive functionalities with the database were implemented using SQL queries. Throughout this process, emphasis was placed on ensuring data integrity by minimizing the possibility of redundancy and anomalies, as well as optimizing the interactive queries.

2. Introduction

Today, the development of various industries, services, and social media has made it possible to purchase and share various products quickly and easily at home. As a result, trends are quickly emerging, being shared, and disappearing among people. This has also influenced the fashion industry, leading to the emergence of the trend known as "Fast Fashion." Fast fashion refers to fashion brands and industries that adopt the latest trends to produce and sell clothing in large quantities at low prices within a short period. The problem is that This trend has encouraged consumers to frequently purchase and throw away clothes carelessly. The production, consumption, and disposal of clothing contribute to environmental pollution, and fast fashion accelerates this process, having a significantly negative impact on the environment[1].

Cotton and polyester, which account for 90% of textile production, each require large amounts of water, pesticides, energy, and chemicals. Additionally, the dyeing process releases wastewater containing heavy metals and toxic substances, adversely affecting the health of animals and local residents. Consumers also discard an average of 36 kilograms of clothing annually, contributing to 5% of landfill waste. While some used clothing is exported to low-income countries instead of being discarded, unsold items are often left abandoned, worsening environmental pollution[2].

This project began with a question about whether the rise of social media and the fast fashion manufacturing and distribution model have truly caused consumers to engage in excessive clothing consumption. While it cannot be denied that changes in people's clothing consumption patterns have been influenced by fast fashion, I believe there is a more fundamental issue at play. Among ordinary people, it’s more common to find those who open their closet one day, feel they have nothing to wear, and buy new clothes, rather than those who collect clothing simply because they like it, even when it’s unnecessary. At this point, we should ask ourselves: Do they truly have nothing to wear? In most cases, the answer is likely no. Because they are not easily visible, hidden in a corner, or simply because we have many clothes but none that seem to match well, we often feel like we don’t have enough clothes. Additionally, when we finally find a favorite piece and decide to wear it after a long time, we might end up throwing it away because it hasn’t been properly cared for. When this cycle repeats, we are left with no choice but to buy new clothes, leading to unnecessary spending and, on a larger scale, accelerating environmental pollution. I believe these issues stem from a lack of proper clothing management. To address this, I designed and developed a wardrobe database to facilitate efficient clothing management.

3. Design and Implementation

The ultimate goal of this project is to reduce clothing waste through efficient wardrobe management, thereby minimizing unnecessary expenses on a small scale and mitigating environmental pollution caused by excessive clothing waste on a larger scale. To achieve this, I developed a database consisting of seven tables and implemented functionalities that allow users to interact with it. Throughout the process, I focused on

efficient data storage, processing, and ensuring data integrity.

3.1 Design

To build a wardrobe database, this project began by designing the database using an ER model, which was then implemented as a relational database. The database consists of the following tables: **Users**, **Clothes**, **Outfits**, **LaundryHistory**, **Tags**, **TagLinking**, and **OutfitClothesLinking**. These tables were utilized to create a database that helps manage clothing efficiently. This section will explain the tables and their attributes, the ER model, and how users can interact with the database.

3.1.1 Tables

The **Users** table is designed to store user information, allowing multiple individuals to use the database. It connects each user to their respective clothing, outfits, and laundry history. The table includes the following attributes: **UserID**, **Name**, **Email**, and **Password**. These attributes are used to distinguish between individual users.

The **Clothes** table is created to store detailed information about clothing items, enabling systematic management. The attributes include **ClothID**, **UserID**, **Type**, **Brand**, **Material**, **Size**, **Color**, **PurchaseDate**, **PurchasePurpose**, **LastWearDate**, **PresentCondition**, and **MaintenanceMethods**. The **ClothID** and **UserID** attributes are included to allow users to uniquely manage specific clothing items. **Type**, **Brand**, **Material**, **Size**, and **Color** are used to store detailed information about the clothing. **PurchaseDate**, **PurchasePurpose**, **LastWearDate**, **PresentCondition**, and **MaintenanceMethods** are included to store information related to the overall management of the clothing items.

The **Outfits** table is designed to store information about outfits created by users by combining various clothing items. It allows users to save their preferred outfits and manage the clothing items included in each outfit. The attributes include **OutfitID**, **UserID**, **OutfitName**, **Season**, **IsFavorite**, and **Notes**. **OutfitID** and **UserID** ensure that users can uniquely manage their outfits. **OutfitName**, **Season**, and **Notes** store detailed information about the outfits, and **IsFavorite** enables users to mark specific outfits as favorites.

The **LaundryHistory** table is designed to manage the washing history of clothing items, helping users preserve and care for their clothes more effectively. The attributes include **LaundryID**, **ClothID**, **LaundryDate**, and **Method**. The **LaundryID** and **ClothID** attributes are included to uniquely manage the washing history of specific clothing items. **LaundryDate** and **Method** are used to store washing-related information.

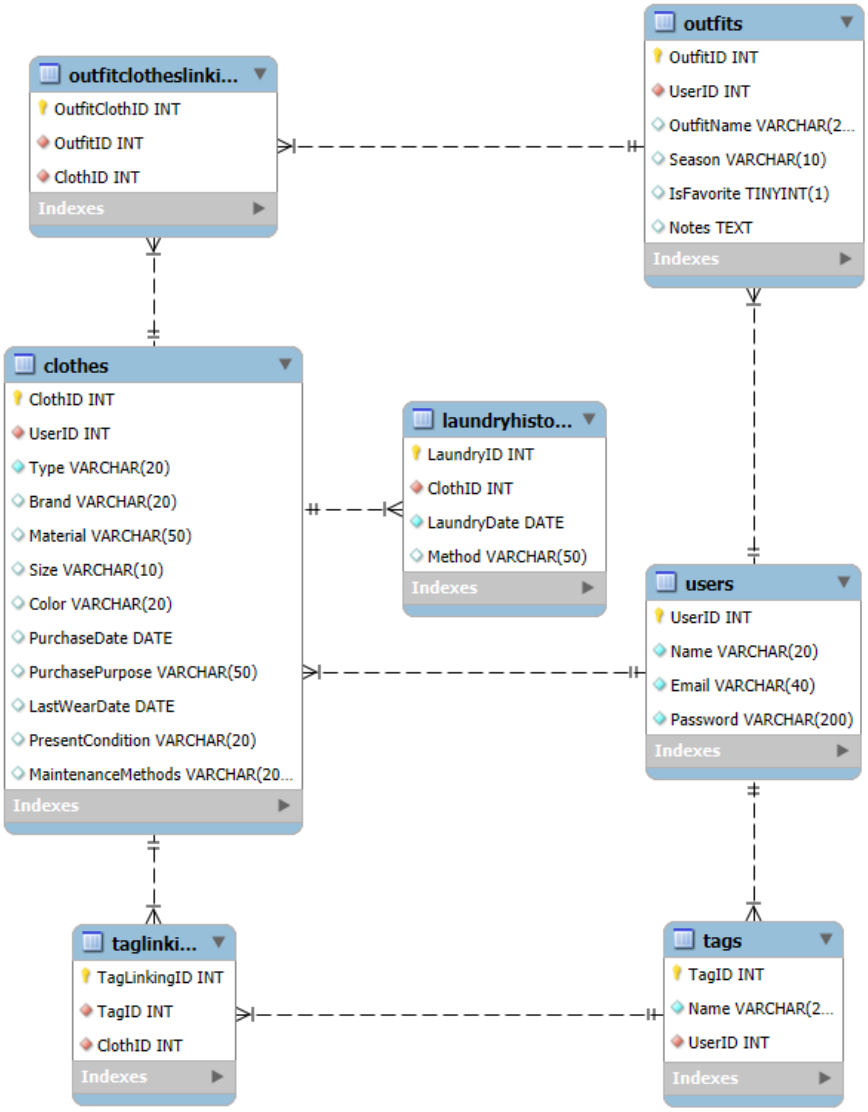
The **Tags** table is designed to enable users to quickly search or categorize clothing items or outfits using tags. The attributes include **TagID**, **Name**, and **UserID**. These attributes are all utilized to allow individuals to set tags, making it easier to manage their clothing items.

The **TagLinking** table is designed to link user-created tags to clothing items. This allows for easy identification of whether a tag has been applied and to which clothing items the same tag has been applied, as well as simplifying the addition, deletion, and modification of tags. The attributes include **TagLinkingID**, **TagID**, and **ClothID**. These attributes are all used to facilitate linking by considering the many-to-many relationship between tags and clothing items.

The **OutfitClothesLinking** table is designed to link user-defined outfits with specific clothing items. The attributes include **OutfitClothID**, **OutfitID**, and **ClothID**. These attributes are used to enable linking by accounting for the many-to-many relationship between outfits and clothing items.

3.1.2 ER - Model

<Figure 1> is a diagram of the ER model used to design the database. It is composed of the tables and attributes explained earlier. The yellow key icons represent primary keys, the red diamond icons denote foreign keys, and the blue diamonds indicate general attributes. The lines in the diagram indicate relationships between entities. A line with a single endpoint represents a "one" relationship, while a line branching into multiple paths indicates a "many" relationship. For instance, examining the line connecting the Users table and the Clothes table reveals that the relationship between Users and Clothes is one-to-many. Additionally, the "Indexes" section contains the indexes set for the table, where the primary keys and foreign keys are configured as indexes.



<Figure 1>ER-model diagram

The relationships between entities can be summarized as follows:

- Users to Clothes: **one-to-many** relationship
  - A user can own multiple pieces of clothing.
- Users to Outfits: **one-to-many** relationship
  - A user can create multiple outfits.
- Users to Tags: **one-to-many** relationship
  - A user can create multiple tags.
- Clothes to LaundryHistory: **one-to-many** relationship
  - A piece of clothing can have multiple laundry records.

- Clothes to OutfitClothesLinking**: one-to-many** relationship
  - A piece of clothing can be included in multiple outfits.
- Clothes to TagLinking: **one-to-many** relationship
  - A piece of clothing can be linked to multiple tags.
- Tags to TagLinking: **one-to-many** relationship
  - A tag can be applied to multiple pieces of clothing.
- Outfits to OutfitClothesLinking: **one-to-many** relationship
  - A single outfit can include multiple pieces of clothing.

This structure is a user-centered data model designed to enable users to systematically manage information about clothing, tags, and outfits, enhancing the user experience by making it easier to handle their own data. Additionally, it establishes relationships between tables to store data efficiently. In particular, the use of intermediary tables (e.g., TagLinking table, OutfitClothesLinking table) for many-to-many relationships (e.g., Tags to Clothes, Outfits to Clothes) ensures that each table can be managed independently, minimizes data redundancy, and improves scalability.

3.1.3 Normalization

The wardrobe database is expected to undergo frequent updates, insertions, and deletions. Therefore, it was normalized to satisfy the Third Normal Form (3NF) to ensure data integrity by minimizing possibility of redundancy and anomalies occurring. 3NF requires that attributes in a table have atomic values, all columns exhibit full functional dependency on the entire primary key, and there is no transitive dependency among non-key attributes. While higher levels of normalization could be applied, it was determined that 3NF is sufficient to meet the database requirements. For instance, although attributes like Brand or Material in the Clothes table might have potential redundancies, managing them as independent entities was considered unnecessary. Separating them would increase the number of joins and complicate data management without significant benefits.

3.1.4 Interacting with Database

Interaction with the database is conducted through queries to process user requests. While the details of these queries will be addressed in the implementation section, this section focuses on the operations performed. Through interaction with the database, users can manage their personal information, clothing items, outfits, laundry records, and tags.

When users save their personal information in the Users table, they are registered in the database and can begin interacting with it. Users can view their stored information or delete it, which will also remove all associated data. By adding and saving their clothing information in the Clothes table, users can later search for items or check their condition to maintain them effectively. Saving outfit combinations created by coordinating different clothing items in the Outfits table allows users to search for outfits based on tags and access their favorites quickly. By storing laundry records in the LaundryHistory table, users can efficiently track the maintenance status of their clothes and even receive alerts for necessary laundering based on the last wash date. Tags can be saved in the Tags table, and their connections to clothing can be established in the TagLinking table using ClothesID and TagID. Similarly, outfits can be linked to clothing in the OutfitClothesLinking table using OutfitID and ClothesID. This enables tag-based searches for both clothing and outfits.

3.2 Implementation

In this project, MySQL Workbench was utilized to implement the database. This section explains the

structure of the database, the SQL queries used to interact with it, and the advantages derived from the implementation methods.

3.2.1 Create Database with Schema and Benefits

In this project, the database was structured with seven tables, as detailed in <Figure 2>. The wardrobe database designed in this way offers several advantages. First, Through "ON DELETE CASCADE," unnecessary related data is automatically deleted, ensuring data integrity. For instance, through this mechanism, when a clothing item is deleted, corresponding laundry records in the LaundryHistory table and tag associations in the TagLinking table are automatically removed. This prevents inconsistencies between data. Second, by minimizing data redundancy and structuring the database with seven tables that consider relationships, storage space is conserved, and data can be managed independently, making maintenance more convenient. Additionally, it is easier to add new attributes or data, enhancing scalability. Third, all reference relationships are clearly defined using foreign keys, ensuring clarity and consistency in data relationships. For example, the UserID in the Clothes table always references the UserID in the Users table, eliminating the possibility of storing invalid UserIDs. This also enhances the connectivity between data. Finally, the use of the UNIQUE constraint prevents data duplication, ensuring data integrity and optimizing storage while maintaining logical consistency. For example, it prevents the same piece of clothing from being added multiple times to the same outfit, thereby eliminating redundancy and avoiding distortion of the outfit's intended meaning.

```
CREATE TABLE `users` (
  `UserID` int NOT NULL AUTO_INCREMENT,
  `Name` varchar(20) NOT NULL,
  `Email` varchar(40) NOT NULL,
  `Password` varchar(200) NOT NULL,
  PRIMARY KEY (`UserID`),
  UNIQUE KEY `Email` (`Email`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb3
```

```
CREATE TABLE `clothes` (
  `ClothID` int NOT NULL AUTO_INCREMENT,
  `UserID` int NOT NULL,
  `Type` varchar(20) NOT NULL,
  `Brand` varchar(20) DEFAULT NULL,
  `Material` varchar(50) DEFAULT NULL,
  `Size` varchar(10) DEFAULT NULL,
  `Color` varchar(20) DEFAULT NULL,
  `PurchaseDate` date DEFAULT NULL,
  `PurchasePurpose` varchar(50) DEFAULT NULL,
  `LastWearDate` date DEFAULT NULL,
  `PresentCondition` varchar(20) DEFAULT NULL,
  `MaintenanceMethods` varchar(200) DEFAULT NULL,
  PRIMARY KEY (`ClothID`),
  KEY `UserID` (`UserID`),
  CONSTRAINT `clothes_ibfk_1` FOREIGN KEY (`UserID`) REFERENCES `users` (`UserID`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3
```

```
CREATE TABLE `outfits` (
  `OutfitID` int NOT NULL AUTO_INCREMENT,
  `UserID` int NOT NULL,
  `OutfitName` varchar(20) DEFAULT NULL,
  `Season` varchar(10) DEFAULT NULL,
  `IsFavorite` tinyint(1) DEFAULT '0',
  `Notes` text,
  PRIMARY KEY (`OutfitID`),
  KEY `UserID` (`UserID`),
  CONSTRAINT `outfits_ibfk_1` FOREIGN KEY (`UserID`) REFERENCES `users` (`UserID`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3
```

```
CREATE TABLE `laundryhistory` (
  `LaundryID` int NOT NULL AUTO_INCREMENT,
  `ClothID` int NOT NULL,
  `LaundryDate` date NOT NULL,
  `Method` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`LaundryID`),
  KEY `ClothID` (`ClothID`),
  CONSTRAINT `laundryhistory_ibfk_1` FOREIGN KEY (`ClothID`) REFERENCES `clothes` (`ClothID`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3
```

```
CREATE TABLE `tags` (
  `TagID` int NOT NULL AUTO_INCREMENT,
  `Name` varchar(20) NOT NULL,
  `UserID` int NOT NULL,
  PRIMARY KEY (`TagID`),
  KEY `UserID` (`UserID`),
  CONSTRAINT `tags_ibfk_1` FOREIGN KEY (`UserID`) REFERENCES `users` (`UserID`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3
```

```
CREATE TABLE `taglinking` (
  `TagLinkingID` int NOT NULL AUTO_INCREMENT,
  `TagID` int NOT NULL,
  `ClothID` int NOT NULL,
  PRIMARY KEY (`TagLinkingID`),
  UNIQUE KEY `TagID` (`TagID`,`ClothID`),
  KEY `ClothID` (`ClothID`),
  CONSTRAINT `taglinking_ibfk_1` FOREIGN KEY (`TagID`) REFERENCES `tags` (`TagID`) ON DELETE CASCADE,
  CONSTRAINT `taglinking_ibfk_2` FOREIGN KEY (`ClothID`) REFERENCES `clothes` (`ClothID`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3

CREATE TABLE `outfitclotheslinking` (
  `OutfitClothID` int NOT NULL AUTO_INCREMENT,
  `OutfitID` int NOT NULL,
  `ClothID` int NOT NULL,
  PRIMARY KEY (`OutfitClothID`),
  UNIQUE KEY `OutfitID` (`OutfitID`,`ClothID`),
  KEY `ClothID` (`ClothID`),
  CONSTRAINT `outfitclotheslinking_ibfk_1` FOREIGN KEY (`OutfitID`) REFERENCES `outfits` (`OutfitID`) ON DELETE CASCADE,
  CONSTRAINT `outfitclotheslinking_ibfk_2` FOREIGN KEY (`ClothID`) REFERENCES `clothes` (`ClothID`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3
```

<figure 2> Create table SQL Queries

3.2.2 Functional Queries and Their Advantages

The database, along with its structure, places equal importance on interaction with users. In this project, various functionalities for interacting with the database were implemented using SQL queries. To ensure these functionalities operate more effectively, several techniques were employed. First, Complex results were simplified by assigning aliases using "AS" to make them more intuitive. Second, as shown in the "3.1.2 ER – Model" section, indexes have been set for each table. These indexes were utilized to enhance search speed. Third, conditions were primarily handled using the WHERE clause, and data was filtered through the WHERE clause before applying the GROUP BY and HAVING clauses. This efficient condition processing minimized the size of the result data and reduced processing time. Finally, DISTINCT was utilized to filter out duplicate data, thereby reducing the result size and optimizing query performance. The following are examples of queries verified to function correctly.

A) User Management

- User Registration

```
INSERT INTO Users(Name, Email, Password)
VALUES('Do Yun', 'vividsu1@naver.com', 'kkk1234')
```

- User Deletion

```
DELETE FROM Users
WHERE UserID =1;
```

- User Information Retrieval

```
SELECT UserID, Name, Email
FROM Users
WHERE UserID=1
```

B) Clothing Management

- Add Clothing

```
INSERT INTO Clothes (UserID, Type, Brand, Material, Size, Color, PurchaseDate, PurchasePurpose, MaintenanceMethods)
VALUES (1, 'Jacket', 'Nike', 'Polyester', 'M', 'Black', '2024-01-01', 'training', 'Avoid Using Fabric Softener');
```

- Search Clothing

```
SELECT ClothID, Type, Brand, Color, PurchaseDate
FROM Clothes
WHERE UserID =1 AND Type ='Jacket';
```

- Update Clothing Status

```
UPDATE Clothes
SET PresentCondition ='Needs Repair', LastWearDate = CURDATE()
WHERE ClothID =1;
```

- Retrieve Clothing Status

```
SELECT ClothID, Type, Color
FROM Clothes
WHERE UserID =1 AND PresentCondition ='Needs Repair';
```

- Analyze Clothing Utilization (Recommend Rarely Worn Clothes)

```sql
SELECT Clothes.ClothID, Clothes.Type, Clothes.LastWearDate, COUNT(LaundryHistory.LaundryID) AS WearCount
FROM Clothes
LEFT JOIN LaundryHistory ON Clothes.ClothID = LaundryHistory.ClothID
WHERE Clothes.UserID =1
GROUP BY Clothes.ClothID
ORDER BY WearCount ASC, Clothes.LastWearDate DESC;
```

C) Outfit Management

- Create Outfit

```sql
INSERT INTO Outfits (UserID, OutfitName, Season, IsFavorite, Notes)
VALUES (1, 'Summer Casual', 'Summer', FALSE, 'Light and comfortable');
```

- Search Outfit

```sql
SELECT DISTINCT Outfits.OutfitID, Outfits.OutfitName, Outfits.Season, Outfits.Notes
FROM Outfits
JOIN OutfitClothesLinking OutfitClothesLinking ON Outfits.OutfitID = OutfitClothesLinking.OutfitID
JOIN TagLinking ON OutfitClothesLinking.ClothID = TagLinking.ClothID
JOIN Tags ON TagLinking.TagID = Tags.TagID
WHERE Tags.Name = 'Casual'
  AND Outfits.UserID = 1;
```

- Set Favorites

```sql
UPDATE Outfits
SET IsFavorite =TRUE
WHERE OutfitID =2;
```

D) Laundry Management

- Add Laundry Record

```sql
INSERT INTO LaundryHistory (ClothID, LaundryDate, Method)
VALUES (1, '2024-11-29', 'Dry Cleaning');
```

- View Laundry History

```sql
SELECT Clothes.ClothID, Clothes.Type, Clothes.Color, MAX(LaundryHistory.LaundryDate) AS LastLaundryDate
FROM Clothes
LEFT JOIN LaundryHistory ON Clothes.ClothID = LaundryHistory.ClothID
WHERE Clothes.UserID =1
GROUP BY Clothes.ClothID
HAVING DATEDIFF(CURDATE(), MAX(LaundryHistory.LaundryDate)) >30;
```

E) Tag Management

- Add Tag

```sql
INSERT INTO Tags (Name, UserID)
VALUES ('Casual', 1);
```

- Search Outfit by Tag

```sql
SELECT Outfits.OutfitID, Outfits.OutfitName, Outfits.ClothIDs, Outfits.Notes
FROM Outfits
JOIN TagLinking ON JSON_CONTAINS(Outfits.ClothIDs, CAST(TagLinking.ClothID AS JSON))
JOIN Tags ON TagLinking.TagID = Tags.TagID
WHERE Tags.Name ='Casual'AND Outfits.UserID =1;
```

- Search Clothing by Tag

```sql
SELECT Clothes.ClothID, Clothes.Type, Clothes.Color
FROM Clothes
JOIN TagLinking ON Clothes.ClothID = TagLinking.ClothID
JOIN Tags ON TagLinking.TagID = Tags.TagID
WHERE Tags.Name ='Casual'AND Clothes.UserID =1;
```

F) TagLinking Management

- Link Tag with Clothing

```sql
INSERT INTO TagLinking (TagID, ClothID) VALUES (1, 1)
```

G) OutfitClothesLinking Management

- Link Outfit with Clothing

```sql
INSERT INTO OutfitClothesLinking (OutfitID, ClothID) VALUES (1, 1)
```

4. Evaluation

All tables in this database are designed to satisfy the Third Normal Form (3NF). As a result, data redundancy is minimized, and anomalies are prevented, ensuring data integrity. This reduces unnecessary storage space wastage caused by duplicate data and simplifies data management. Additionally, when creating tables, all reference relationships are explicitly defined using foreign keys, thereby maintaining referential integrity and strengthening integrity constraints through the use of options such as ON DELETE CASCADE and UNIQUE constraints. When creating queries to implement functionalities, techniques such as the appropriate use of DISTINCT, WHERE clauses, and indexes are employed to optimize query performance and reduce processing time.

One potential drawback is that, since the tables are normalized and divided, the number of JOIN increases, and additional queries may be required. This can make queries more complex, complicate maintenance, and increase processing time. Additionally, when inserting or updating data, all related tables need to be manipulated, increasing the workload and making the process more cumbersome.

5. Conclusion

This project aimed to prevent unnecessary spending on clothing and, furthermore, to reduce wasted clothing to help protect the environment. To achieve this, a wardrobe database was created to manage clothing, and interactive functionalities were implemented using SQL queries. By utilizing this database, clothing can be managed more efficiently, which will lead to a gradual reduction in people's excessive clothing consumption. This, in turn, may influence the supply patterns of fast fashion brands, reducing their acts of overproduction. Ultimately, this will contribute to achieving the fundamental goal behind the creation of this database. Currently, the focus is on database implementation, and the service is not yet ready for practical use. Therefore, the next tasks will include developing an application to make the functionalities created with queries usable in practice, adding more diverse features, and continuously optimizing the database.

Reference

1. 김보미, et al., *패스트 패션에 의한 환경오염: 현황 및 전망.* 대한환경공학회지, 2023. **45**(11): p. 506-518.
2. Bick, R., E. Halsey, and C.C. Ekenga, *The global environmental injustice of fast fashion.* Environmental Health, 2018. **17**: p. 1-4.