# Warehouse Management System

**Database Management System**

**MSCS 542L-256-24F**

**TechBoys**


Marist College
School of Computer Science and Mathematics

Submitted To:
Dr. Reza Sadeghi

Oct 30, 2024

# PROJECT REPORT OF WAREHOUSE MANAGEMENT SYSTEM

**Team Name**
TechBoys

**Team Members**

| | |
|---|---|
| Sumanth Kumar Katapally | SumanthKumar.Katapally1@marist.edu (Team Head) |
| Abhijeet Cherungottil | Abhijeet.Cherungottil1@marist.edu (Team Member) |
| Sagar Shankaran | Sagar.Shankaran1@marist.edu (Team Member) |

# TABLE OF CONTENTS

# TABLE OF FIGURES

# 1. DESCRIPTION OF TEAM MEMBERS

**Sumanth Kumar Katapally:**

I attained my Bachelor's degree in Computer Science from Keshav Memorial Institute of Technology in 2022. I kickstarted my career as a Software Developer Intern at Virtusa, after which I took up a full-time role of a Software Developer at DBS TECH thereby gaining an overall experience of 2 years and 5 months. I am proficient in Java, Spring, Spring Boot, Angular, MongoDB and MySQL. I came to Marist College to pursue my MS in Computer Science concentrating in Artificial Intelligence.

**Abhijeet Cherungottil:**

I am a passionate computer science graduate currently pursuing an MS in Cloud Computing at Marist College, set to graduate in the 2026 batch. I also have 3 years of hands-on experience in iPhone app development, with expertise in Swift 5 and Swift UI. My current team is from India, which creates familiarity within the group and allows us to communicate easily with each other. I chose Sumanth because he mentioned having previous experience with GitHub and actively volunteered.

**Sagar Shankaran:**

I am a 2026 batch student of Masters in AI program at Marist College. I have experience in full stack development for 3 years. I have worked across various projects including MERN stack and Android Apps. I have chosen this team because all the members have a good understanding of each other and have profound knowledge of the project. Abhijeet and Sumanth have good relevant skills.

# 2. INTRODUCTION

In today's dynamic retail environment, efficient warehouse management is crucial for ensuring smooth operations, especially when multiple shopkeepers rely on a shared space to store and manage their inventory. The Warehouse Management System (WMS) project is designed to meet these needs by providing a secure, organized, and user-friendly platform that enhances the management of a warehouse where shopkeepers store and haul their cloth boxes under the oversight of an administrative user.

The WMS aims to streamline warehouse operations by offering a robust set of features tailored to both admin and shopkeeper needs. Admins will benefit from secure login capabilities, enabling them to manage access by adding or removing shopkeepers and adjusting credentials as needed. The system will also empower admins to oversee the entire inventory, allowing them to add, edit, or delete items with detailed information such as item type, storage time, ID, name, quantity, price, and storage location. Additionally, admins will have the authority to review and manage borrowing requests, ensuring that warehouse resources are allocated efficiently.

For shopkeepers, the WMS offers intuitive features designed to simplify their interactions with the warehouse. They will be able to search for items based on various attributes, save a list of favorite items, and request to borrow or purchase items for specific time periods. The system will also keep a detailed history of borrowed items, providing transparency and accountability.

User experience is at the forefront of the WMS design, with a welcoming interface, clear navigation, and tabular reports to present organized lists of requested items. Security is also a key focus, with options for password encryption and safeguards against potential errors like duplicate entries or exceeding borrowing limits. This WMS will revolutionize warehouse management, making it more efficient and secure for all users involved.

# 3. PROJECT OBJECTIVE

The objective of this project is to design and implement a robust Warehouse Management System (WMS) that provides an organized, secure, and user-friendly platform for managing a warehouse where multiple shopkeepers store and haul their cloth boxes, overseen by an admin user. The WMS will offer comprehensive functionalities, including:

1. **Admin Management:**
   - Secure login for the admin with the ability to change credentials.
   - Add and remove shopkeepers by creating or deleting their usernames and passwords.
   - Manage the warehouse inventory by adding, editing, and deleting items, including details such as item type, storage time, ID, name, quantities, price, and storage location.
   - Review and manage borrowing requests, with the ability to accept or reject them.
2. **Shopkeeper Features:**
   - Search for items in the warehouse based on various attributes like ID, name, and producer.
   - Save a list of favorite items for quick reference.
   - Request to borrow or purchase items for specific periods.
   - View the history of borrowed items.
3. **User Experience:**
   - Provide a welcoming page and a clear menu of functions on all pages.
   - Display reports in a tabular format, showing organized lists of requested items.
   - Include an exit function that thanks users for using the software.

- Implement warnings for specific actions, such as duplicate item IDs, exceeding borrowing limits, or null search results.

4. **Security:**
   - Protect user information with optional encryption of passwords.

# 4. REVIEW

### 5.1 Zoho Inventory [1]:

- Helps with creating and tracking the inventory (stock flow) of Items and Item Groups.
- Integrates with popular e-commerce platforms and monitors your stock flow across multiple sales channels.
- Creates sales orders, raises invoices, gets paid instantly by integrating into a payment gateway, prints package slips and generates shipment labels.
- Helps in adding custom fields to your orders, invoices, bills, and payment receipts.

### 5.2 Fishbowl Inventory [2]:

- Provides visibility and control of inventory across all locations.
- Integrates with accounting systems like Intuit QuickBooks Online, Xero, and Reckon Accounts to provide real-time inventory and accounting reporting.
- Allows users to scan items by barcode, product number, UPC, or SKU to update inventory records, make accounting adjustments, and customize prices.
- Includes features like auto reorder points, in-depth reporting, and the ability to create purchase orders, sales orders, and pick tickets.
- Helps businesses track inventory spending, find ways to save money on purchasing, and avoid over-ordering.

### 5.3 Korber Warehouse Management Systems [3]:

- Integrates with other supply chain solutions, such as yard management and transportation, and ERP systems.

- Is flexible and adaptable, with configuration functionality that reduces the need for extensive customization and development.
- Is available as a software-as-a-service (SaaS), which includes provisioning, maintenance, and support.
- Has over 99% inventory accuracy and 80-85% less inventory loss.

# 5. MERITS

1. Add a notification to alert the user to order more items when the count reaches 'n'

- WMS proactively alerts users (admin or shopkeepers) when the inventory level of a particular item falls below a specified threshold (n), ensuring that the stock levels are replenished before they run out.
- In older or manual systems, stock monitoring is typically done manually. Users might only realize an item is out of stock when it's too late, leading to delays in fulfilling orders.

2. Tracks the location of items within the warehouse, including specific aisles, racks, and bins.

- WMS offers detailed location tracking for each item in the warehouse, down to the aisle, rack, and bin level, ensuring that every item can be quickly located.
- In older systems or warehouses, item locations may be tracked in a general sense, such as by section or area, or even on paper. This can lead to inefficiencies in locating items.

3. Ensuring old stock is cleared before new stock is added.

- Many traditional systems don’t enforce stock rotation, meaning newer stock can be placed over older stock, increasing the risk of product damage

4. Manages items that are out of stock and ensures they are prioritized when they become available.

- Out-of-stock items may not be tracked effectively in less advanced systems, which can result in delays in fulfilling backorders or not promptly notifying users when the items become available

5. Adding sustainability tracking.
    - Less advanced WMSs rarely integrate sustainability tracking. Most inventory systems focus purely on operational aspects like stock levels, ignoring environmental or sustainability impacts.

# 6. GITHUB REPOSITORY

https://github.com/Sumanthkatapally/DBMS_PROJECT.git

# 7. ENTITY RELATIONSHIP MODEL (ER MODEL)

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects, or concepts relate to each other within a system. ER Diagrams are most often used to design debug relational databases in the fields of software engineering, business information systems, education, and research.

| Entities | Attributes |
|---|---|
| **ADMIN** | AdminID, Username, Password, Role, Email |
| **SHOPKEEPER** | ShopkeeperID, Username, Password, FullName, Email |
| **PRODUCT** | productid, productName, CategoryId, Size, PricePerUnit, QuantityInStock |
| **WAREHOUSE** | WarehouseID, LocationId, MaxCapacity, CurrentCapacity, AvailableSpace |
| **BORROWREQUEST** | RequestID ,RequestDate,BorrowStatus,ProductID ,ShopkeeperID |
| **CATEGORY** | CategoryID ,CategoryName,Description,Createddate,Seasonality |
| **SUPPLIER** | SupplierID ,SupplierName,ContactPerson,Email,SuppliedCloths |
| **ORDER** | orderId, orderDate, ShopkeeperID, Orderstatus, Amount |

| | |
|---|---|
| **LOCATION** | locationId, Address, PhoneNumber, Manager, OpeningHours, TotalCapacity |
| **TRANSACTION** | TransactionID, OrderID, TransactionDate, PaymentMethod, Amount, DiscountApplied, FinalAmount, Invoice |
| **STOCKAUDIT** | AuditID, WarehouseID, AuditDate, AuditorName, Discrepancies |
| **SHIPMENT** | ShipmentID, OrderID, ShipmentDate, ShipmentStatus, TrackingNumber |

# 7.1 IMPLEMENTATION OF ER DIAGRAM

The entity-relationship conceptual model of a database can be represented graphically, using a diagram. These diagrams are simple and intuitive, and their basic components are:

• Rectangles - specifying entity types

• Ellipses - specifying attributes

• Rhomboids - specifying relationship types

• Lines - connections between entities and attributes, or entities and relationship types.

In the picture, you can see a more detailed overview of the notation:

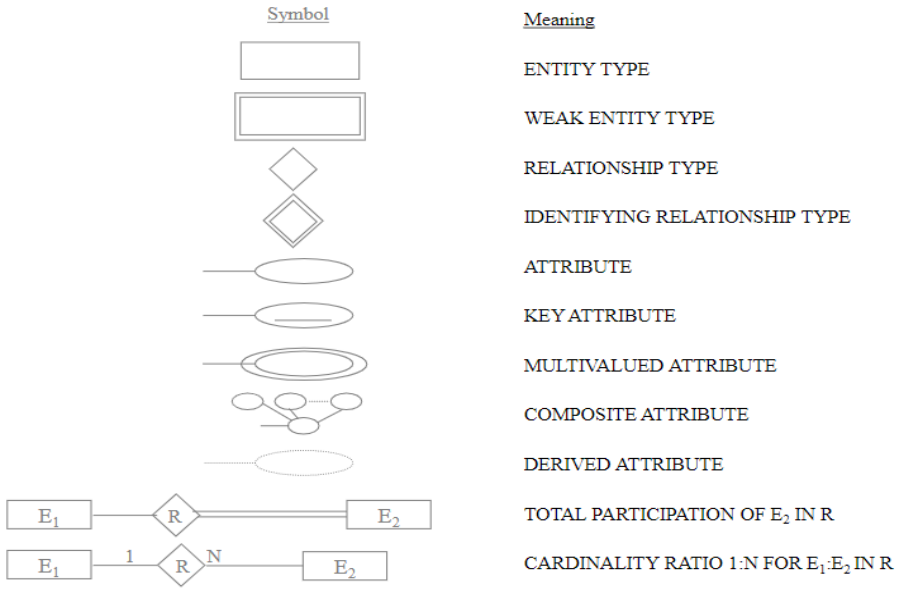



Figure 7. 1 Relationship of Entity Relationship Model

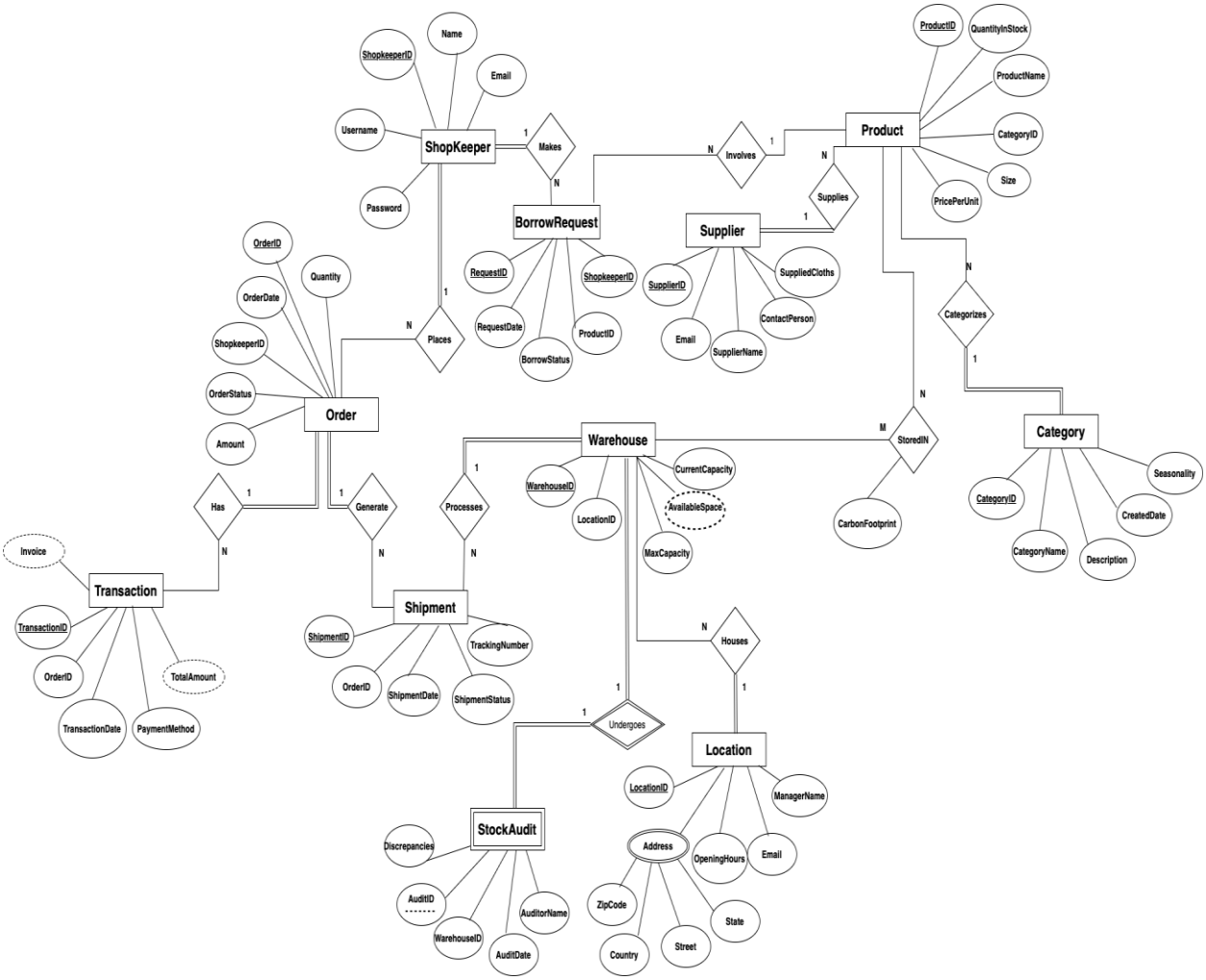



Figure 7.2 External ER SHOPKEEPER POV [6]

As an external ER user, as a customer, I might interact with the Product and Category entities when browsing items. The Order and Shipment entities would be relevant when making a purchase and tracking my delivery.

The Warehouse and Product entities suggest that I might be able to check item availability across different locations, which could be useful and the presence of the Shipment entity with attributes like TrackingNumber and ShipmentStatus indicates I could likely track my orders easily.

The Product entity seems to contain detailed information (like Size, PricePerUnit), which would be helpful for making purchasing decisions.

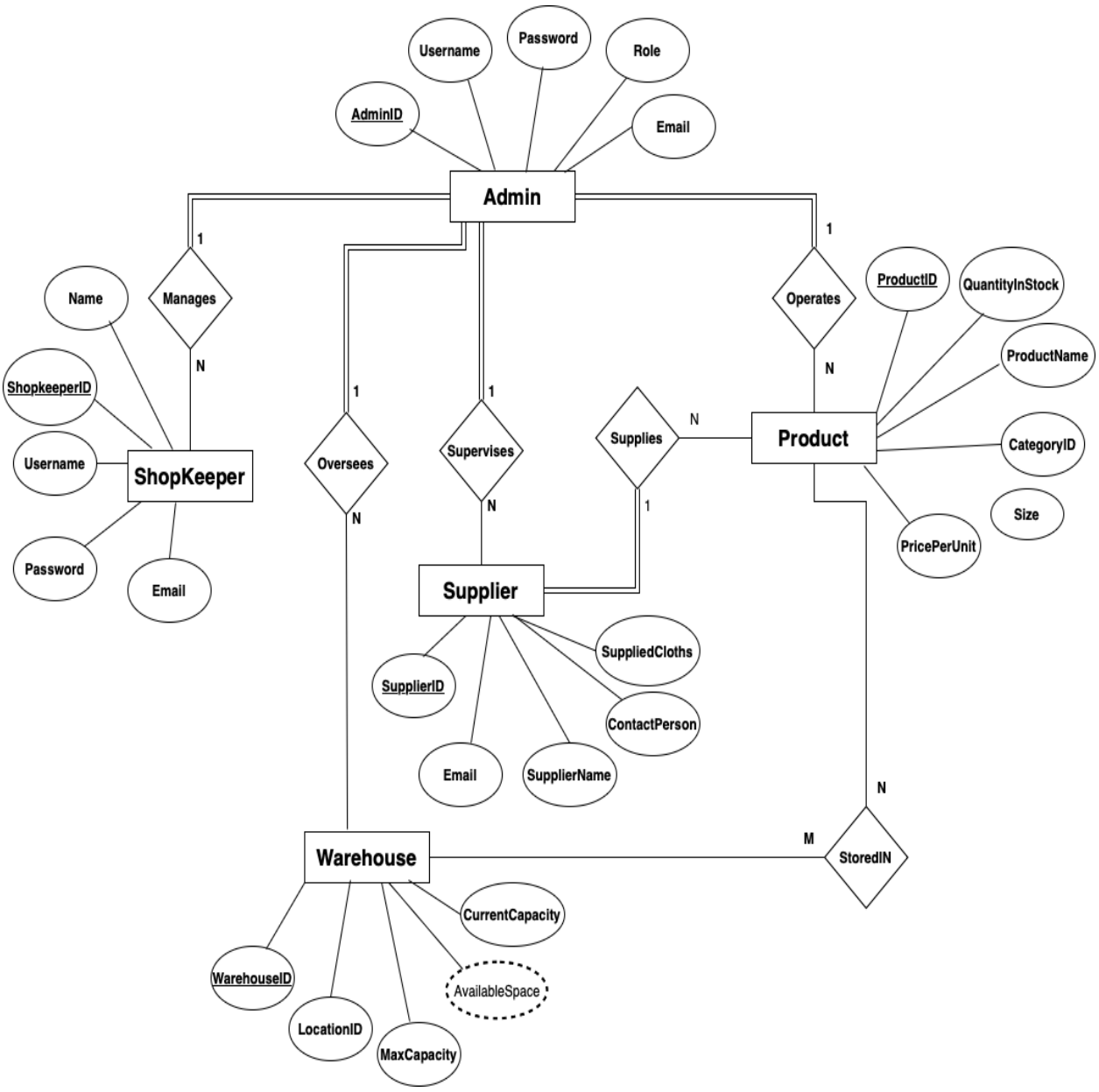



Figure 7. 3 External ER ADMIN POV [6]

The diagram places the Admin entity at the center, indicating a centralized management approach. The Admin has oversight of ShopKeepers, Suppliers, and Products, implying broad control over the entire system. This centralized structure allows for efficient top-down management and decision-making processes.

Product management appears to be a core focus of the system. Products are linked to Suppliers who provide them, Warehouses where they are stored, and are managed by both Admins and ShopKeepers. This comprehensive approach to product tracking enables detailed inventory control and supply chain visibility.

Warehouse management is given particular attention, with attributes like CurrentCapacity, MaxCapacity, and AvailableSpace. The inclusion of AvailableSpace as a potentially derived attribute (shown in a dashed oval) suggests dynamic capacity management, crucial for efficient inventory control and storage optimization.

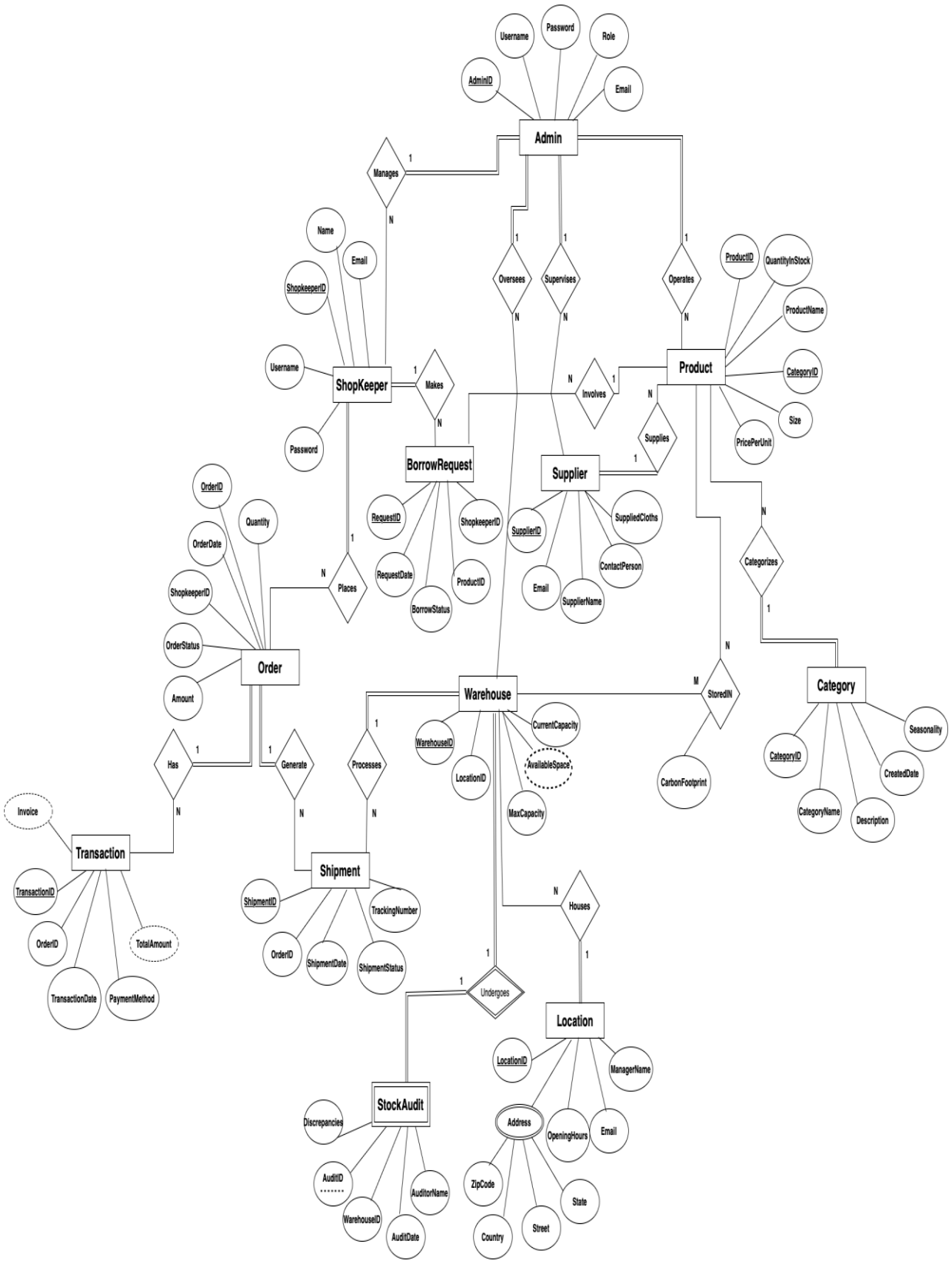



Figure 7. 4 Entity Relationship Diagram [6]

# 7.2 ENHANCED ENTITY RELATIONSHIP MODEL (EER MODEL)

EER Diagram, also abbreviated as Enhanced Entity-relationship diagram, helps us create and maintain detailed databases through high-level models and tools. In addition, they are developed on the basic ER diagrams and are its extended version.

We have used MySQL Workbench to create the EER diagram. Enhanced Entity Relationship (EER) diagrams are an essential part of the modeling interface in MySQL Workbench. EER diagrams provide a visual representation of the relationships among the tables in our model.

EER diagrams extend ER diagrams by introducing the concepts of **superclasses** and **subclasses** to depict inheritance between entities. Specialization and generalization allow for categorizing entities into more specific subtypes or abstracting them into a generalized form. **Aggregation** represents a higher-level abstraction where relationships themselves are treated as entities.

EER Diagrams basically help in creating and maintaining excellent databases with the help of smart and efficient techniques. It is a visual representation of the plan or the overall outlook of the database you intend to create.

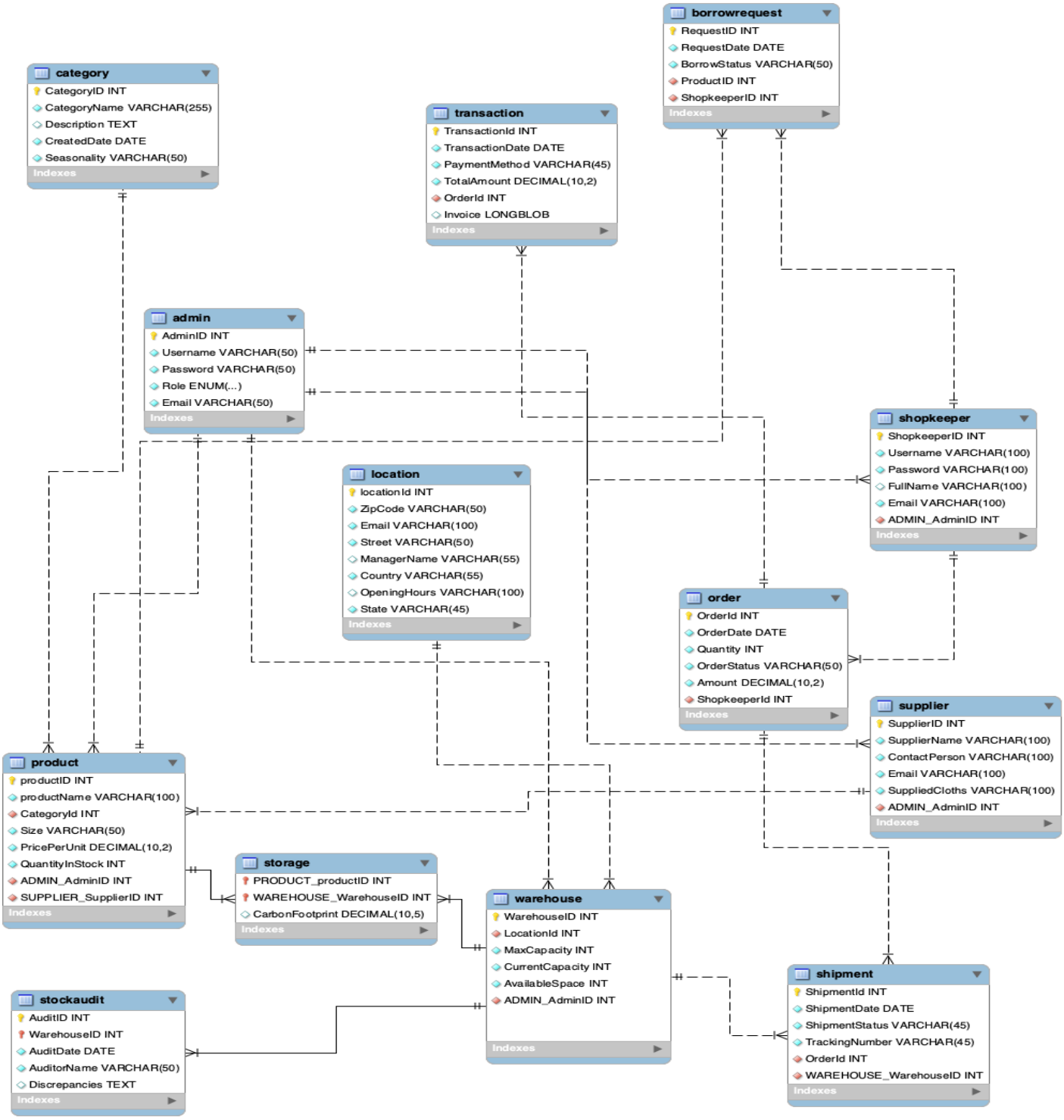



Figure 7.5 Enhanced Entity Relationship

## Entities and Attributes [4][5]:

**Entity:Admin**

**Description:** administrator responsible for managing warehouses and overseeing operations.

**Attributes:**

**AdminID (PK):** Unique identifier for each admin

**Username:** Login name for the admin

**Password:** Encrypted password for account security

**Role:** Specifies the admin's role (e.g., Manager, Supervisor)

**Email:** Contact email for the admin

**Entity: SHOPKEEPER**

**Description:** Represents store managers who place orders and manage inventory.

**Attributes:**

**ShopkeeperID (PK):** Unique identifier for each shopkeeper

**Username:** Login name for the shopkeeper

**Password:** Encrypted password for account security

**FullName:** Composite attribute (FirstName, LastName)

**Email:** Contact email for the shopkeeper

**Entity: PRODUCT**

**Description:.** Represents items available for sale in the clothing retail system

**Attributes:**

**productID (PK):** Unique identifier for each product

**productName: Name of the product**

**CategoryId (FK): References the category the product belongs to**

**Size: Size of the product (e.g., Small, Medium, Large)**

**PricePerUnit: Cost of a single unit of the product**

**QuantityInStock: Current available quantity of the product**

**Entity: WAREHOUSE**

**Description:** Represents storage facilities for products.

**Attributes:**

**WarehouseID (PK):** Unique identifier for each warehouse

**LocationId (FK):** References the location of the warehouse

**MaxCapacity:** Maximum storage capacity of the warehouse

**CurrentCapacity:** Current occupied capacity (derived)

**AvailableSpace:** Remaining storage space (derived)

**Entity: BORROWREQUEST**

**Description:** Represents requests from shopkeepers to borrow products.

**Attributes:**

**RequestID (PK):** Unique identifier for each request

**RequestDate:** Date when the request was made

**BorrowStatus:** Current status of the request (e.g., Pending, Approved, Rejected)

**ProductID (FK):** References the product being requested

**ShopkeeperID (FK):** References the shopkeeper making the request

**Entity: CATEGORY**

**Description:** Represents different classifications of products.

**Attributes:**

**CategoryID (PK):** Unique identifier for each category

**CategoryName:** Name of the category (e.g., Casual, Formal, Sportswear)

**Description:** Detailed description of the category

**Createddate:** Date when the category was created

**Seasonality:** Applicable season(s) for the category

**Entity: SUPPLIER**

**Description:** Represents companies or individuals who supply products.

**Attributes:**

**SupplierID (PK):** Unique identifier for each supplier

**SupplierName:** Name of the supplier company or individual

**ContactPerson:** Composite attribute (FirstName, LastName) of the primary contact

**Email:** Contact email for the supplier

**SuppliedCloths:** Multivalued attribute listing supplied product IDs

**Entity: ORDER**

**Description:** Represents customer purchases or stock orders.

**Attributes:**

**orderId (PK):** Unique identifier for each order

**orderDate:** Date when the order was placed

**ShopkeeperID (FK):** References the shopkeeper who placed the order

**Orderstatus:** Current status of the order (e.g., Processing, Shipped, Delivered)

**Amount:** Total cost of the order

**Entity: LOCATION**

**Description:** Represents physical locations of warehouses or shops.

**Attributes:**

**locationId (PK):** Unique identifier for each location

**Address:** Composite attribute (Street, City, State, ZipCode)

**PhoneNumber:** Contact number for the location

**Manager:** Composite attribute (FirstName, LastName) of the location manager

**OpeningHours:** Multivalued attribute listing opening hours for each day

**TotalCapacity:** Sum of all warehouse capacities at this location (derived)

**Entity: TRANSACTION**

**Description:** Represents financial transactions related to orders.

**Attributes:**

**TransactionID (PK):** Unique identifier for each transaction

**OrderID (FK):** References the associated order

**TransactionDate:** Date when the transaction occurred

**PaymentMethod:** Method used for payment (e.g., Cash, Credit Card)

**TotalAmount:** Total amount of the transaction

**Invoice:** BLOB

**Entity: STOCKAUDIT**

**Description:** Represents periodic checks of warehouse inventory accuracy.

**Attributes:**

**AuditID (Partial Key):** Identifier for the audit, unique within a warehouse

**WarehouseID (PK, FK):** References the warehouse being audited, part of the primary key

**AuditDate:** Date when the audit was conducted

**AuditorName:** Composite attribute (FirstName, LastName) of the auditor

**Discrepancies:** Multivalued attribute listing product IDs and quantity differences

**Entity: SHIPMENT**

**Description:** Represents the process of delivering orders to customers.

**Attributes:**

**ShipmentID (PK):** Unique identifier for each shipment

**OrderID (FK):** References the associated order

**ShipmentDate:** Date when the shipment was sent

**ShipmentStatus:** Current status of the shipment (e.g., Preparing, In Transit, Delivered)

**TrackingNumber:** Unique number for tracking the shipment

**Multivalued Attributes:**

- SUPPLIER: SuppliedCloths (list of supplied product IDs)
- LOCATION: OpeningHours (list of opening hours for each day)
- STOCKAUDIT: Discrepancies (list of product IDs and quantity differences)

**Composite Attributes:**

- SHOPKEEPER: FullName (FirstName, LastName)
- SUPPLIER: ContactPerson (FirstName, LastName)
- LOCATION: Address (Street, City, State, ZipCode)
- LOCATION: Manager (FirstName, LastName)
- STOCKAUDIT: AuditorName (FirstName, LastName)

**Derived Attributes:**

- WAREHOUSE: CurrentCapacity (derived from sum of stored products)
- WAREHOUSE: AvailableSpace (derived from MaxCapacity - CurrentCapacity)
- LOCATION: TotalCapacity (derived from sum of all warehouse capacities at this location)
- TRANSACTION: AMOUNT(derived from quantity, price per unit from order table)

**Weak Entity:**

- STOCKAUDIT (its existence depends on WAREHOUSE)

**Strong Entities:**

ADMIN, SHOPKEEPER, PRODUCT, WAREHOUSE, BORROWREQUEST, CATEGORY, SUPPLIER, ORDER, LOCATION, TRANSACTION, SHIPMENT

**Total Participations:**

- ORDER to TRANSACTION (every order must have at least one transaction)
- ORDER to SHIPMENT (every order must have at least one shipment)
- WAREHOUSE to STOCKAUDIT (every warehouse must undergo stock audits)

**Partial Participations:**

- PRODUCT to BORROWREQUEST (not all products may be involved in borrow requests)
- SHOPKEEPER to BORROWREQUEST (not all shopkeepers may make borrow requests)

**Cardinality Ratios:**

**a. One-to-One (1:1):**

- WAREHOUSE to STOCKAUDIT (one warehouse undergoes one stock audit)

**b. One-to-Many (1:N):**

- ADMIN to WAREHOUSE (one admin can manage many warehouse)
- SHOPKEEPER to ORDER (one shopkeeper can place many orders)
- CATEGORY to PRODUCT (one category can have many products)
- SUPPLIER to PRODUCT (one supplier can supply many products)
- ORDER to TRANSACTION (one order can have many transactions)
- LOCATION to WAREHOUSE (one location can house many warehouses)
- ORDER to SHIPMENT (one order can have many shipments)
- WAREHOUSE to SHIPMENT (one warehouse can process many shipments)

c. **Many-to-Many (M:N):**

- PRODUCT to WAREHOUSE (many products can be stored in many warehouses)

## 8. PRESENTATION SUBJECT

**Data Types** specify the type of data that a column can contain. String Data Types specify that the column can hold data made up of characters.[7][8]

**String Data Types:**

1. **CHAR**: Creates a fixed length that one can declare after creating the table [9]
    - Its length value can be between 0 and 255.
    - If one skips declaring the length variable, it is automatically assigned to 1.
    - If the user enters a string that is shorter than the declared length, the database will pad the remaining space with spaces to reach the specified length.
    - Ex: SELECT fName + char(20) + lastname as Ename FROM employees
2. **VARCHAR:** Allows a varying length for the inputted string [8]
    - Length ranges from 0 to 65,535
    - VARCHAR inputs are not padded: varchar(24) would only specify that the column can hold a maximum of 24 characters. 20 characters would be stored as 20 characters, not 24 (20 + 4 padding)
    - This can reduce the amount of storage required to save data
    - Ex: CREATE TABLE Human {firstname VARCHAR(24),lastname VARCHAR(24)};
3. **BINARY**: refers to any data that is stored in a format of 1s and 0s, typically representing non-text content such as images, videos, audio, or any other file formats.[10]

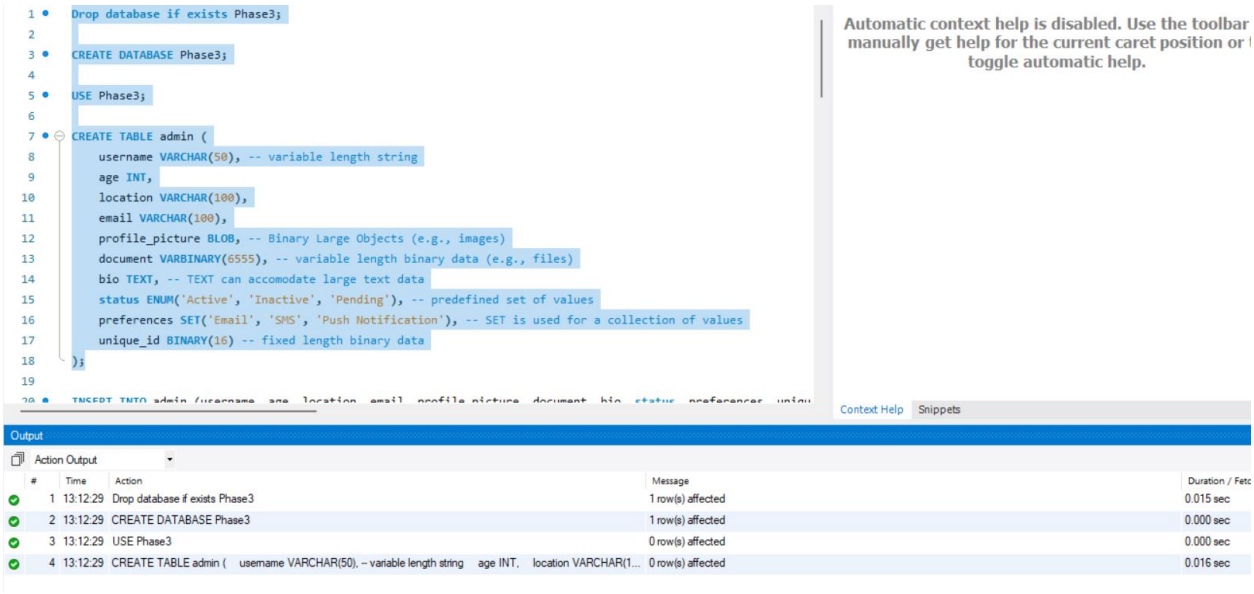
- Stores a set string of bytes.
- Similar to CHAR, except it stores binary strings instead of nonbinary strings.
- The binary column is a fixed length column.
- CREATE TABLE Star{number BINARY};

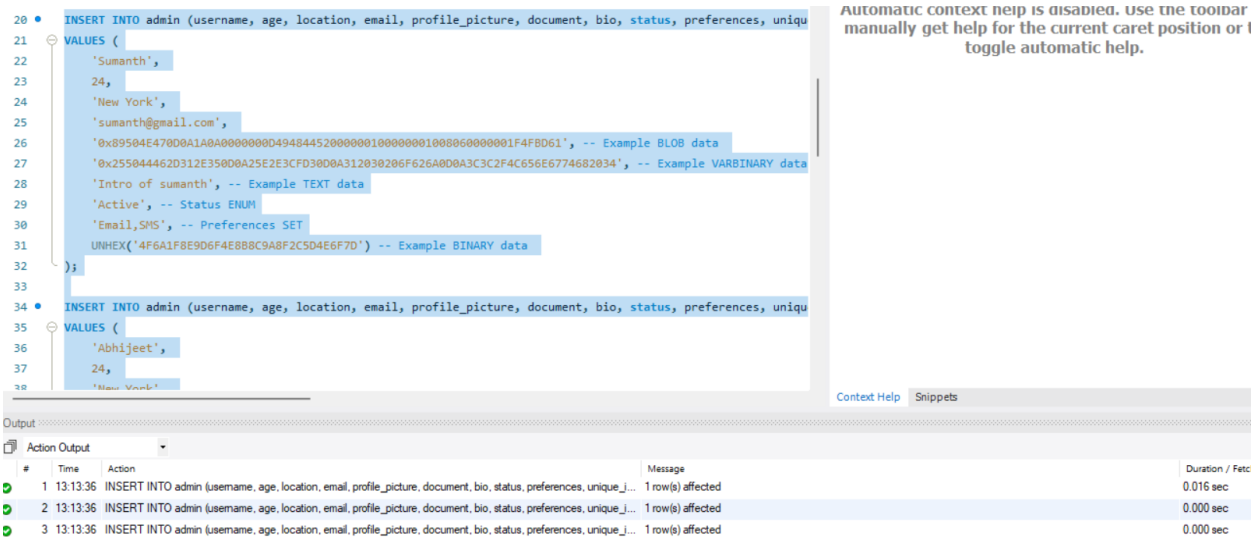
4. **VARBINARY**: Variable length binary data.[10]
    - Similar to VARCHAR, except it stores binary strings instead of nonbinary strings.
    - Unlike CHAR and VARCHAR, the length of BINARY and VARBINARY is measured in byes but not in characters.
    - CREATE TABLE Star{radius VARBINARY(4)};
5. **BLOB:** A binary large object that can hold a variable amount of data[11]
    - Four blob types: tinyblob, blob, mediumblob and longblob.
    - They differ only in the maximum length of values they can hold.
    - Blob values are treated as binary strings(byte strings), they have binary character.
    - CREATE TABLE List{description BLOB};
6. **TEXT:** Stores any kind of text data. Can contain both single-byte and multibyte characters.
    - Text values are treated as nonbinary strings (character strings), they have a character set other than binary.
    - There are four types: tinytext, text, mediumtext and longtext.
    - CREATE TABLE List{description TEXT};
7. **ENUM:** A string object with a value chosen from a list of permitted values that are enumerated explicitly in the column specification table at creation time. [12]
    - The strings you specify as input values are automatically encoded as numbers.
    - The numbers are translated back to the corresponding strings in query results.
    - Enum can have a maximum of 65,535 distinct elements.

8. **SET:** A string object that can have zero or more values each of which must be chosen from a list of permitted values specified when the table is created .[13]
    - Set column values that consist of multiple set members must be separated by columns.
    - A set column can have a maximum of 64 distinct members.
    - CREATE TABLE game{statics SET('a','b','c')};

**String Data Type Examples:**

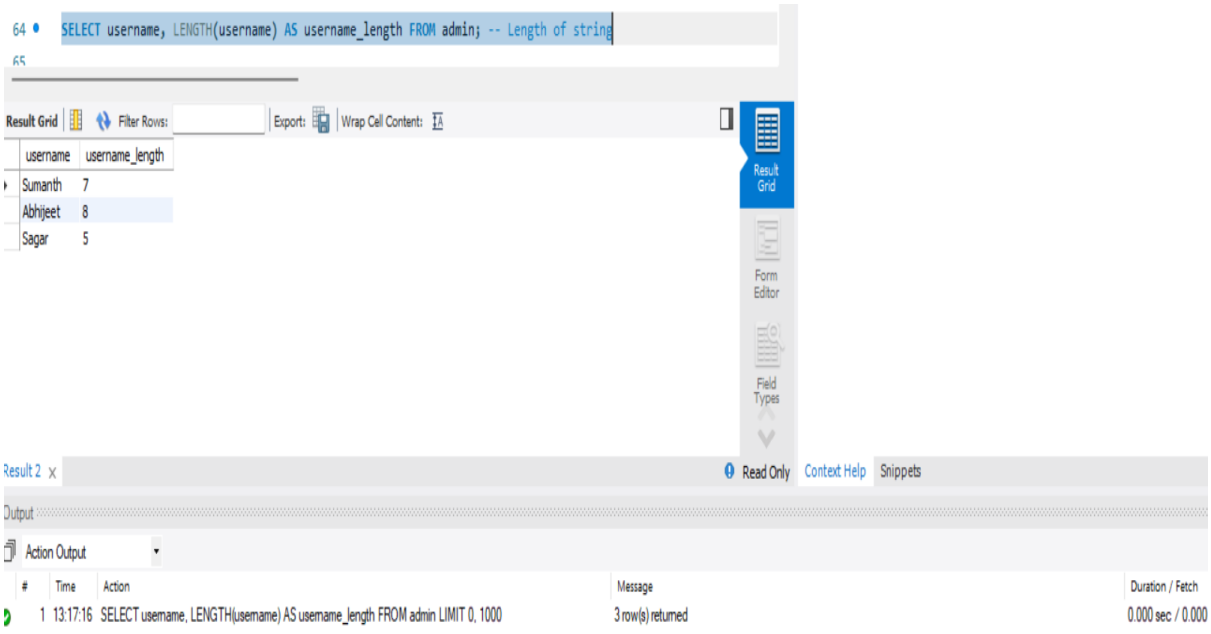
1. **Table creation and Insertion:**



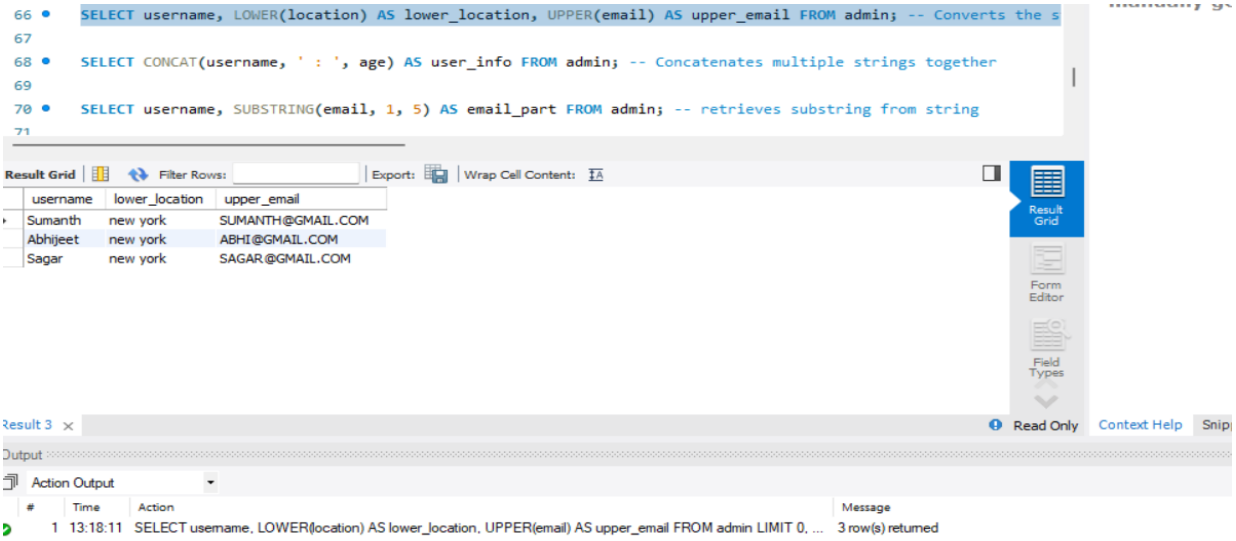



## String Functions:

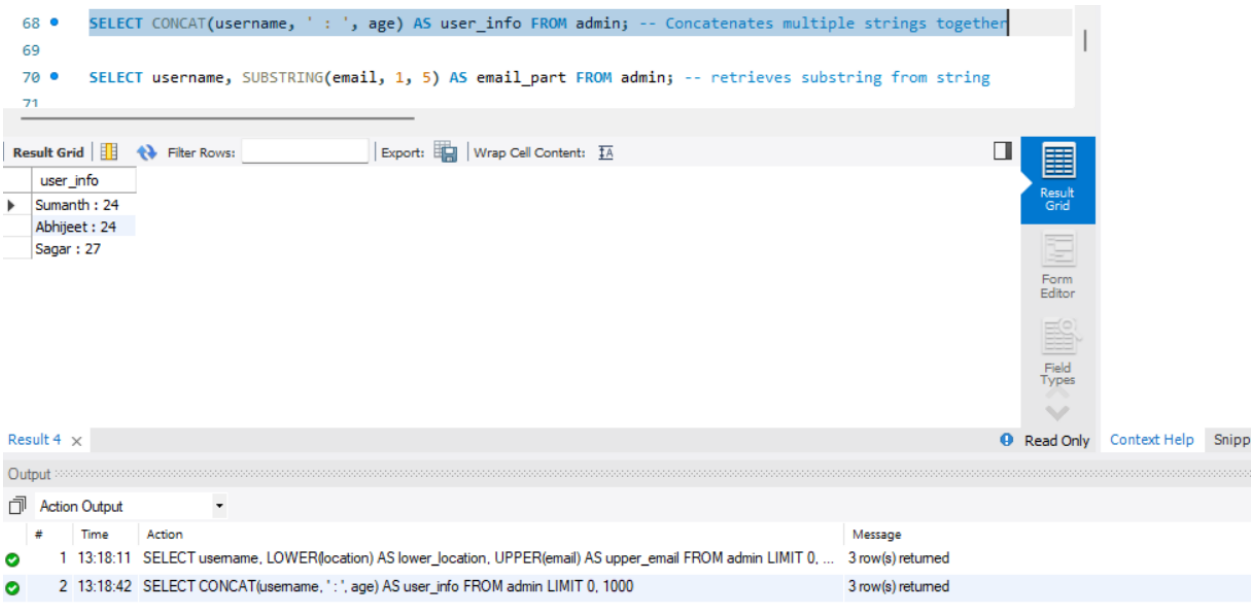
1. **<u>LENGTH ( ) :</u>** Provides length of string



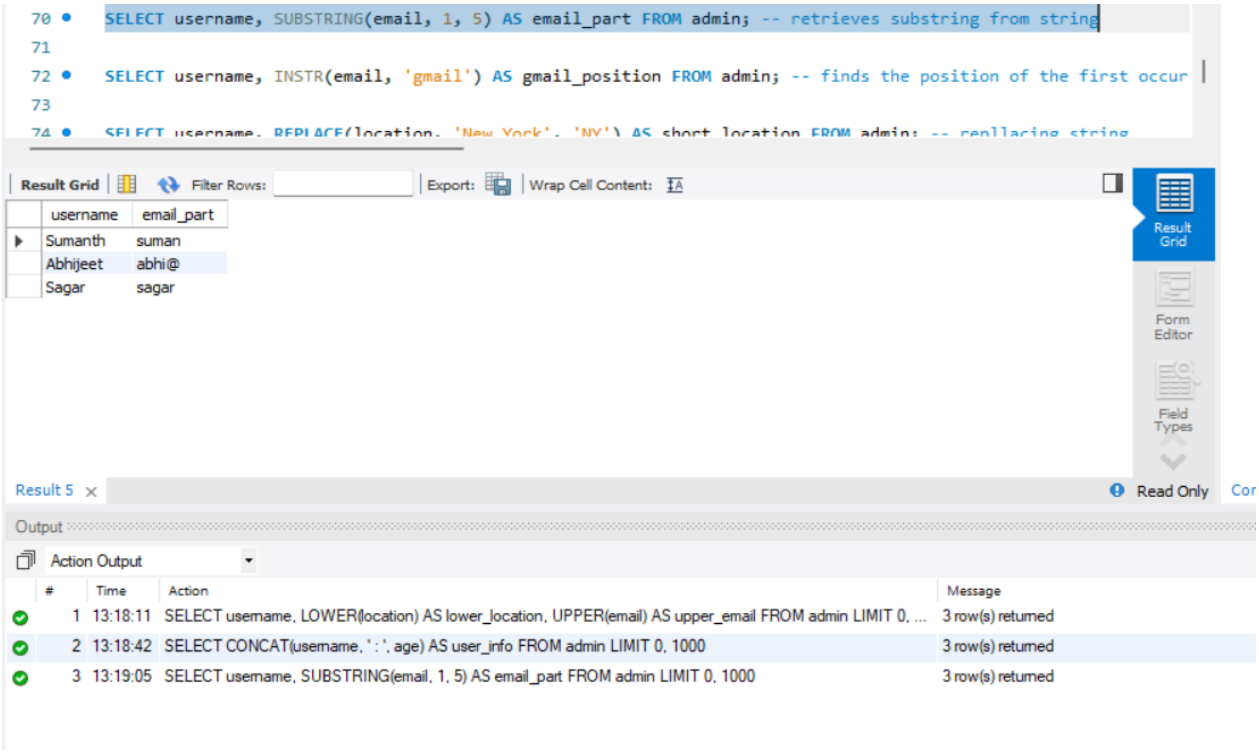
2. **Lower () & Upper ():** Converts the string to lower and upper case



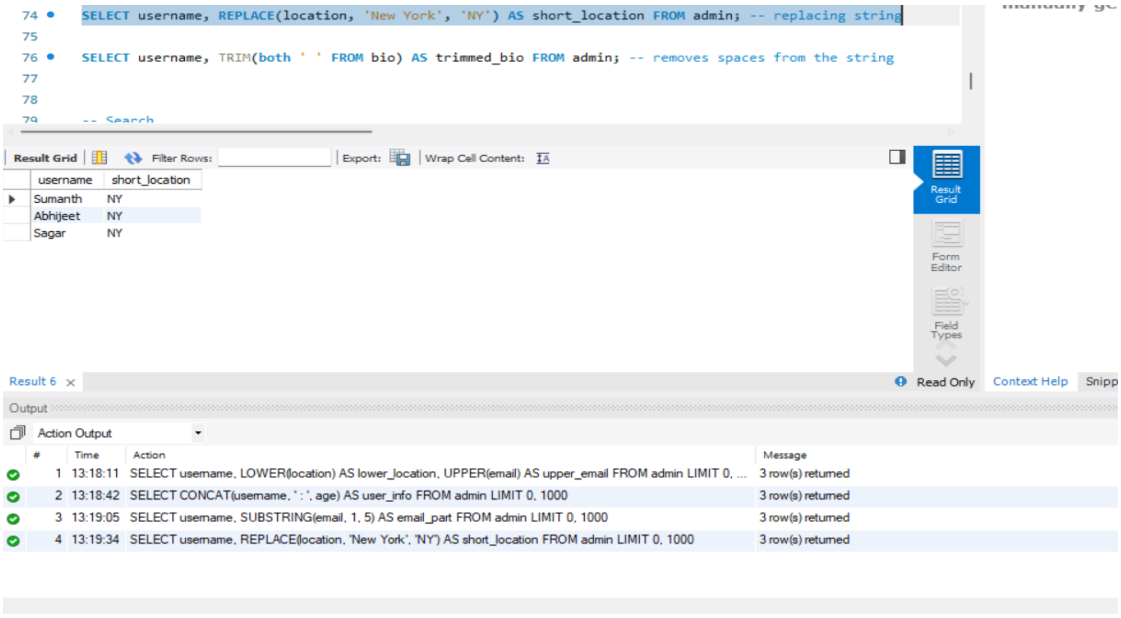

3. **Concat:** Concatenates multiple strings together



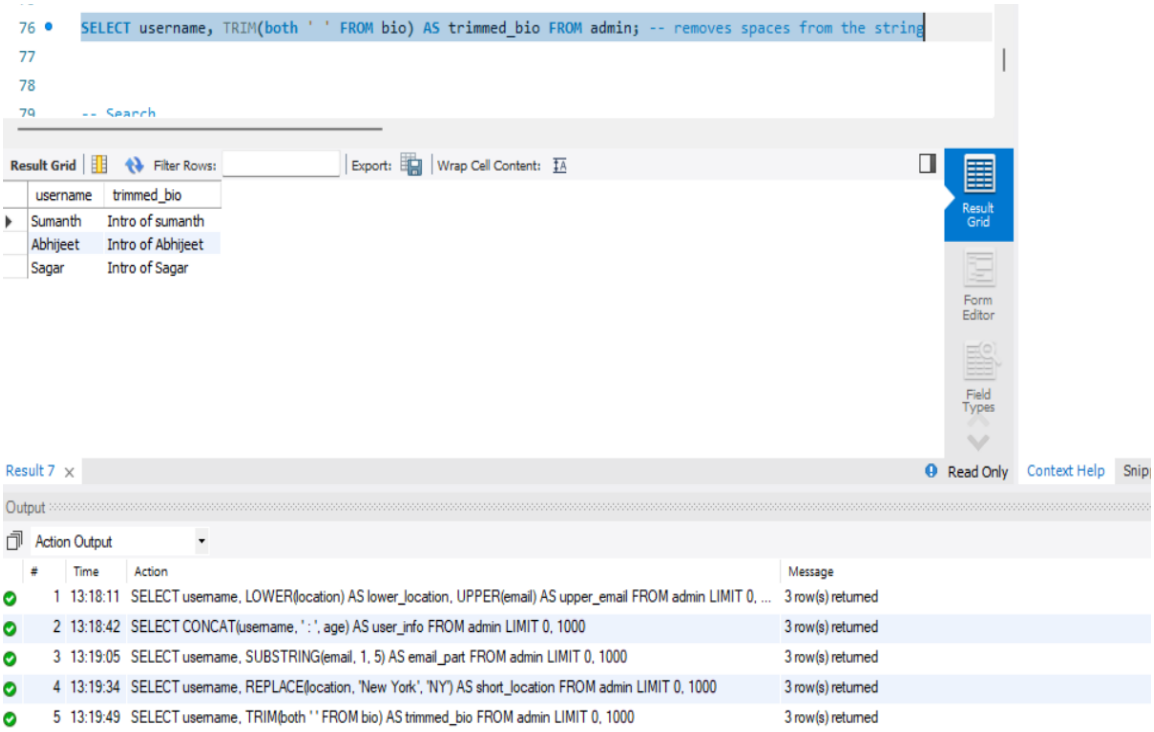
4. **Substring ():** Retrieves substring from string



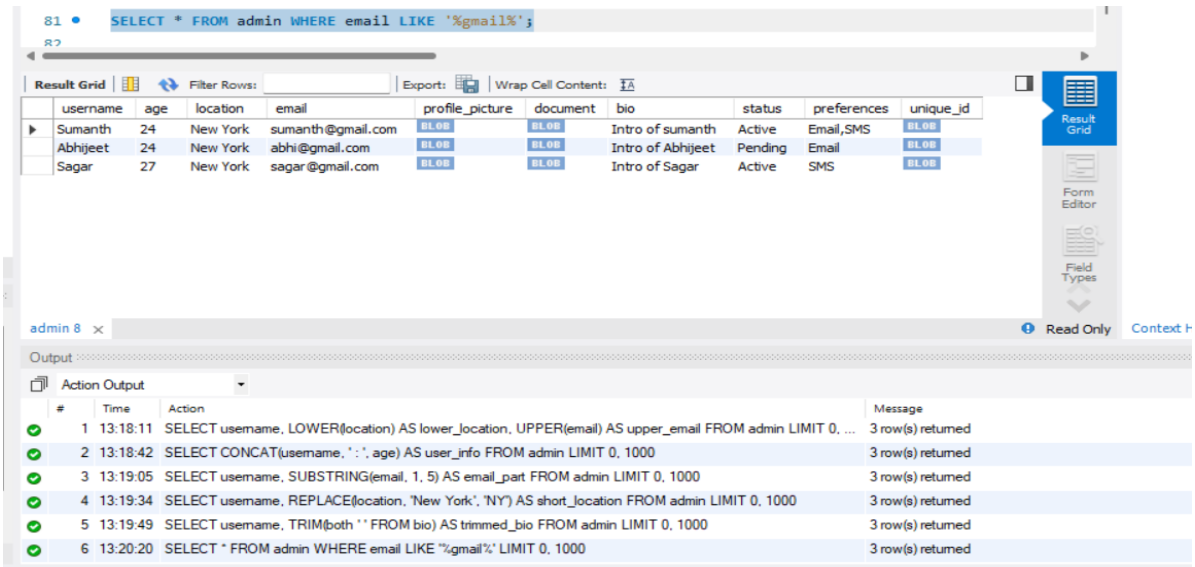

5. **Replace ():** Replaces string with provided input string.

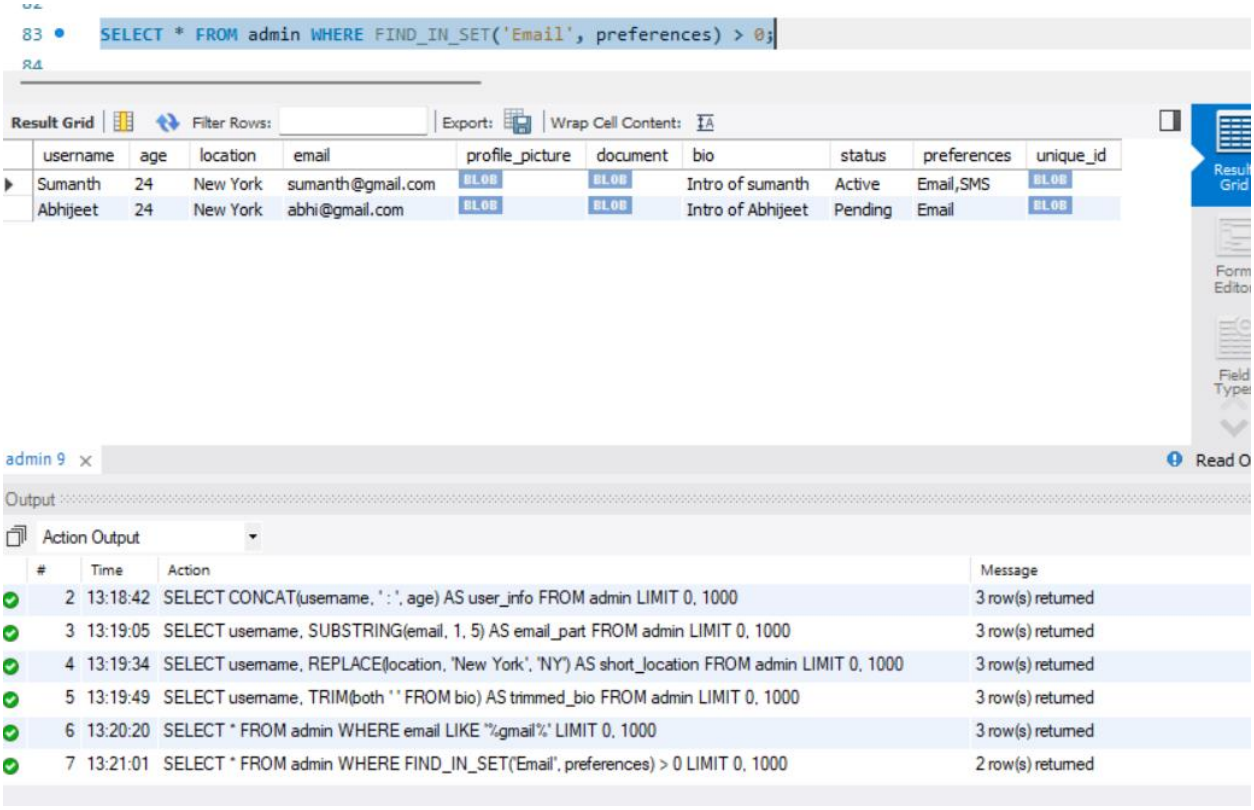


6. **TRIM ():** Removes spaces from the string

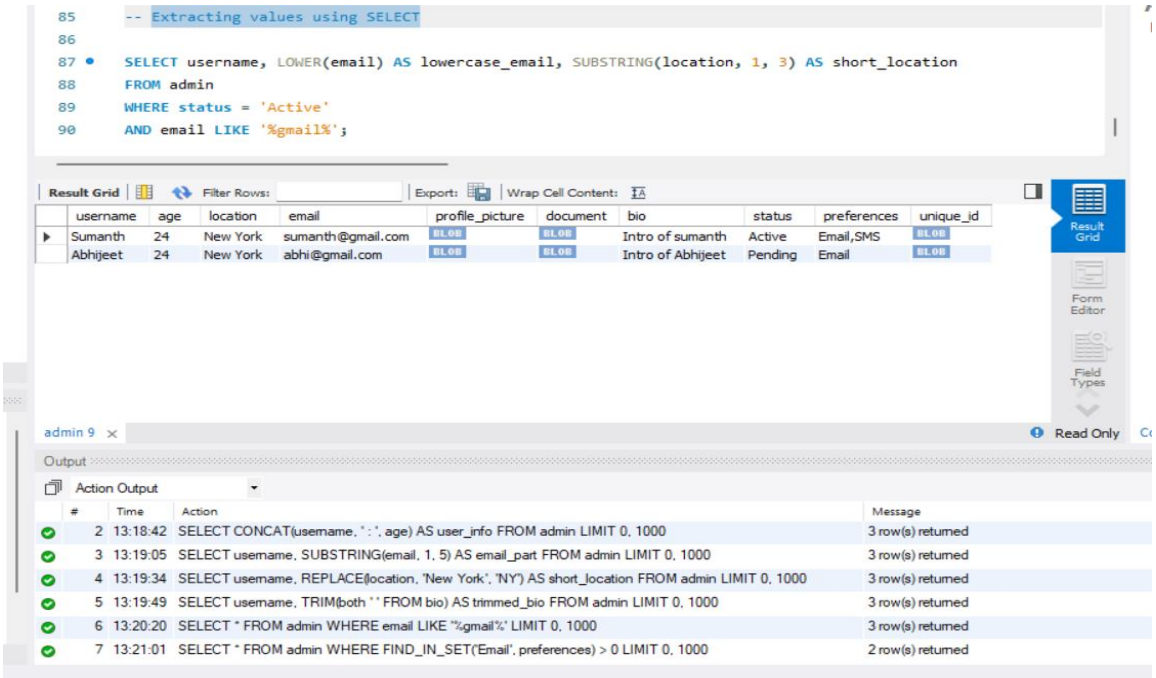



7. **Searching:**






## 8. Extracting values using SELECT:



## JSON Data Types:[16]

JSON is a lightweight, text-based data format designed to be easy for humans to read and write, and easy for machines to parse and generate.

Introduced in MySQL 5.7.8 for better support of structured and unstructured data.

Often used for APIs and storing complex data like arrays, objects, etc.

1. **Creation & Insertion:**

   Storing structured data as a JSON column allows you to store complex data types.

- Create and insertion of a data to table with a JSON column

```
13    -- Inserting JSON data type values into the table
14
15    -- Direct Insertion of JSON object
16    INSERT INTO json_demo_table(json_data, description)
17    VALUES ('{"key1":"value1","key2":"value2"}', 'Direct JSON object insertion');
18
19
20    INSERT INTO json_demo_table(json_data, description)
21    VALUES ('[3, "string1"]', 'Direct JSON array insertion');
22
23    -- JSON object insertion using JSON_OBJECT() function
24    INSERT INTO json_demo_table(json_data, description)
25    VALUES (JSON_OBJECT("key3", "value3", "key4", "value4"), 'Insertion using JSON_OBJECT()');
```

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 14:40:22 | CREATE TABLE json_demo_table ( json_data JSON, description VARCHAR(100) ) | 0 row(s) affected |
| 2 | 14:40:57 | INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1":"value1","key2":"value2"}', 'Direct J... | 1 row(s) affected |
| 3 | 14:40:57 | INSERT INTO json_demo_table(json_data, description) VALUES ('[3, "string1"]', 'Direct JSON array insertion') | 1 row(s) affected |

- Insertion using functions

```
23    -- JSON object insertion using JSON_OBJECT() function
24    INSERT INTO json_demo_table(json_data, description)
25    VALUES (JSON_OBJECT("key3", "value3", "key4", "value4"), 'Insertion using JSON_OBJECT()');
26
27    -- JSON array insertion using JSON_ARRAY() function
28    INSERT INTO json_demo_table(json_data, description)
29    VALUES (JSON_ARRAY("string2", 3, '{"key5":"value5","key6":"value6"}'), 'Insertion using JSON_ARRAY()');
30
31    -- Array inside JSON object
32    INSERT INTO json_demo_table(json_data, description)
33    VALUES ('{"key1":["string3", 6], "key2":"value2"}', 'Nested JSON: array in object');
34
35    -- Display the contents of the table
36    SELECT * FROM json_demo_table;
37
38
39    -- Merging JSON documents
40
41    SELECT JSON_MERGE_PRESERVE(json_data, '[1, 2]') JSON_MERGE_PRESERVE('[1, 2]', json_data) JSON_MERGE_PRESERVE(json_data, '{
```

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 14:40:22 | CREATE TABLE json_demo_table ( json_data JSON, description VARCHAR(100) ) | 0 row(s) affected |
| 2 | 14:40:57 | INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1":"value1","key2":"value2"}', 'Direct J... | 1 row(s) affected |
| 3 | 14:40:57 | INSERT INTO json_demo_table(json_data, description) VALUES ('[3, "string1"]', 'Direct JSON array insertion') | 1 row(s) affected |
| 4 | 14:41:43 | INSERT INTO json_demo_table(json_data, description) VALUES (JSON_OBJECT("key3", "value3", "key4", "va... | 1 row(s) affected |
| 5 | 14:41:44 | INSERT INTO json_demo_table(json_data, description) VALUES (JSON_ARRAY("string2", 3, '{"key5":"value5",... | 1 row(s) affected |

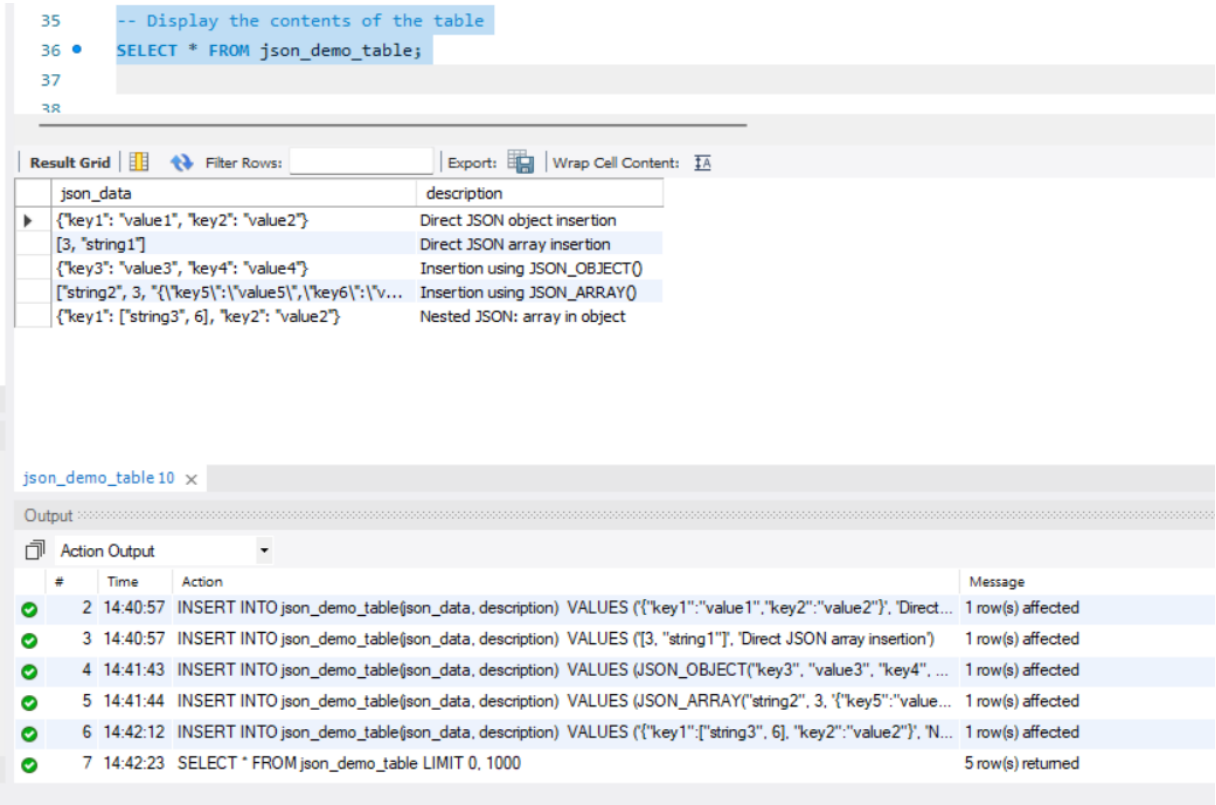
2. Using array in JSON object:

```
31    -- Array inside JSON object
32    INSERT INTO json_demo_table(json_data, description)
33    VALUES ('{"key1":["string3", 6], "key2":"value2"}', 'Nested JSON: array in object');
34
35    -- Display the contents of the table
36    SELECT * FROM json_demo_table;
37
38
```

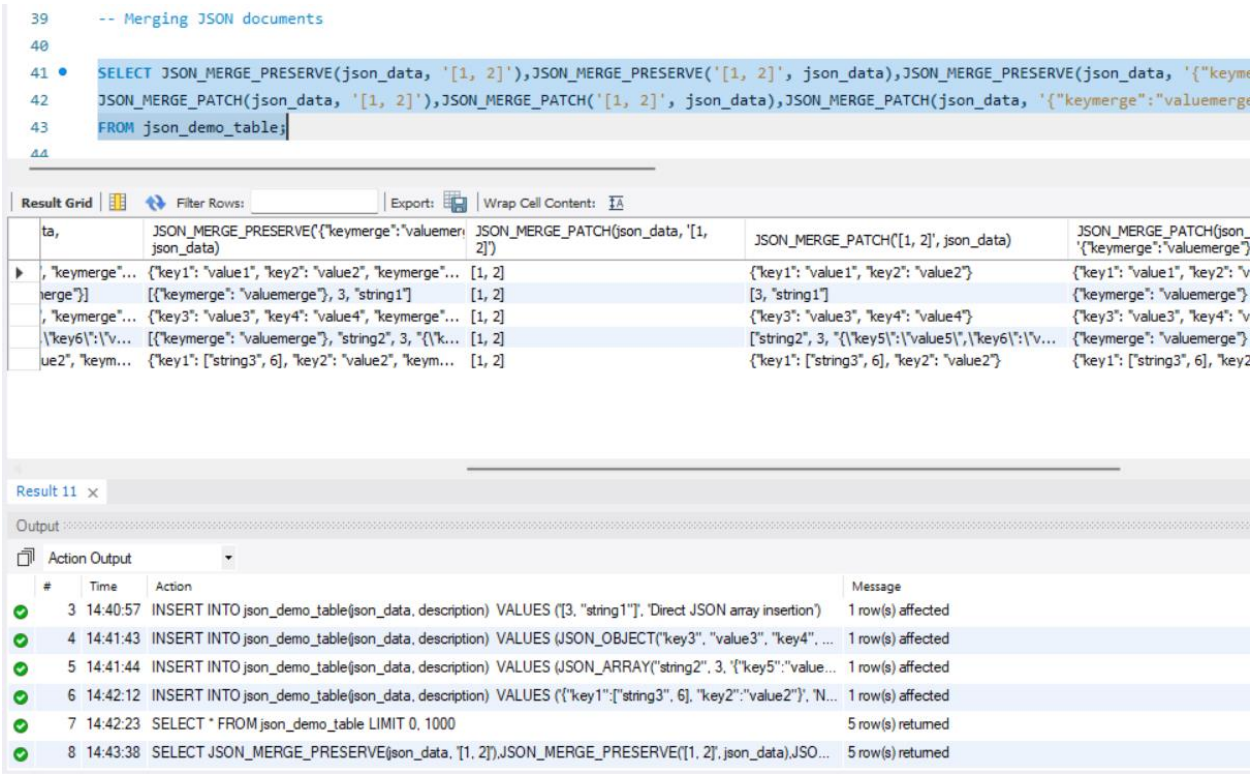
Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 2 | 14:40:57 | INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1":"value1","key2":"value2"}', 'Direct... | 1 row(s) affected |
| 3 | 14:40:57 | INSERT INTO json_demo_table(json_data, description) VALUES ('[3, "string1"]', 'Direct JSON array insertion') | 1 row(s) affected |
| 4 | 14:41:43 | INSERT INTO json_demo_table(json_data, description) VALUES (JSON_OBJECT("key3", "value3", "key4", ... | 1 row(s) affected |
| 5 | 14:41:44 | INSERT INTO json_demo_table(json_data, description) VALUES (JSON_ARRAY("string2", 3, '{"key5":"value... | 1 row(s) affected |
| 6 | 14:42:12 | INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1":["string3", 6], "key2":"value2"}', 'N... | 1 row(s) affected |

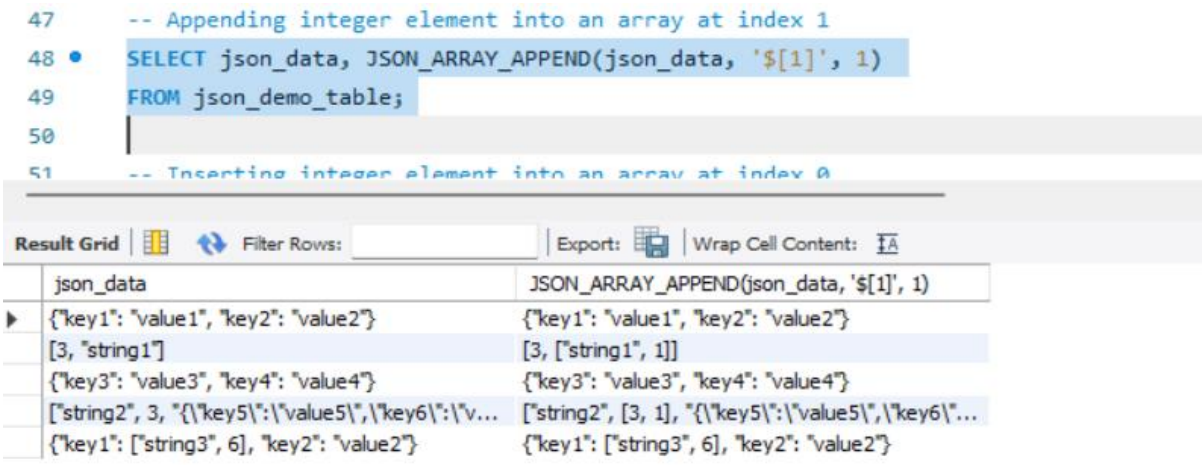



3. Merge:
    - JSON_MERGE_PRESERVE( ) Merging JSON data type columns with JSON array and vice versa
    - JSON_MERGE_PATCH( ) Merging JSON data type columns with JSON array and vice versa




## JSON Functions:

### 1. JSON_ARRAY_APPEND ():



## 2. JSON_ARRAY_INSERT():

```
51      -- Inserting integer element into an array at index 0
52 •    SELECT JSON_ARRAY_INSERT(json_data, '$[0]', 1)
53      FROM json_demo_table;
54
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| JSON_ARRAY_INSERT(json_data, '$[0]', 1) |
|---|
| {"key1": "value1", "key2": "value2"} |
| [1, 3, "string1"] |
| {"key3": "value3", "key4": "value4"} |
| [1, "string2", 3, "{\"key5\":\"value5\",\"key6\":\... |
| {"key1": ["string3", 6], "key2": "value2"} |

Result 13

Output

Action Output

| # | Time | Action |
|---|---|---|
| 5 | 14:41:44 | INSERT INTO json_demo_table(json_data, description) VALUES (JSON_ARRAY("string2", 3, '{"key5":"value... |
| 6 | 14:42:12 | INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1":["string3", 6], "key2":"value2"}', 'N... |
| 7 | 14:42:23 | SELECT * FROM json_demo_table LIMIT 0, 1000 |
| 8 | 14:43:38 | SELECT JSON_MERGE_PRESERVE(json_data, '[1, 2]'),JSON_MERGE_PRESERVE('[1, 2]', json_data),JSO... |
| 9 | 14:44:31 | SELECT json_data, JSON_ARRAY_APPEND(json_data, '$[1]', 1) FROM json_demo_table LIMIT 0, 1000 |
| 10 | 14:45:29 | SELECT JSON_ARRAY_INSERT(json_data, '$[0]', 1) FROM json_demo_table LIMIT 0, 1000 |

## 3. JSON_STORAGE_SIZE ():

```
55      -- Calculating the storage size of JSON objects
56 •    SELECT json_data, JSON_STORAGE_SIZE(json_data)
57      FROM json_demo_table;
58
59      -- Determining the type of JSON elements
60 •    SELECT json_data, JSON_TYPE(json_data)
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| json_data | JSON_STORAGE_SIZE(json_data) |
|---|---|
| {"key1": "value1", "key2": "value2"} | 41 |
| [3, "string1"] | 19 |
| {"key3": "value3", "key4": "value4"} | 41 |
| ["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v... | 56 |
| {"key1": ["string3", 6], "key2": "value2"} | 52 |

## 4. JSON_TYPE ():

```
59      -- Determining the type of JSON elements
60      SELECT json_data, JSON_TYPE(json_data)
61      FROM json_demo_table;
62
63      -- Getting the length of JSON elements (array length or number of key-value pair
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| json_data | JSON_TYPE(json_data) |
|---|---|
| {"key1": "value1", "key2": "value2"} | OBJECT |
| [3, "string1"] | ARRAY |
| {"key3": "value3", "key4": "value4"} | OBJECT |
| ["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v... | ARRAY |
| {"key1": ["string3", 6], "key2": "value2"} | OBJECT |

Result 15 ×

Output

## 5. JSON_LENGTH():

```
63      -- Getting the length of JSON elements (array length or number of key-value pairs in object)
64      SELECT json_data, JSON_LENGTH(json_data)
65      FROM json_demo_table;
66
67      -- Validating JSON objects (returns 1 if valid, 0 otherwise)
68      SELECT JSON_VALID('string'), JSON_VALID('"string1"');
69
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| json_data | JSON_LENGTH(json_data) |
|---|---|
| {"key1": "value1", "key2": "value2"} | 2 |
| [3, "string1"] | 2 |
| {"key3": "value3", "key4": "value4"} | 2 |
| ["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v... | 3 |
| {"key1": ["string3", 6], "key2": "value2"} | 2 |

Result 16 ×

Output

6. **JSON_VALID:** Validating JSON objects (returns 1 if valid, 0 otherwise)

```
67    -- Validating JSON objects (returns 1 if valid, 0 otherwise)
68 •  SELECT JSON_VALID('string'), JSON_VALID('"string1"');
69
70    -- Search and selection within JSON documents
71
72    -- Using JSON_EXTRACT() to search for values by key
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| JSON_VALID('string') | JSON_VALID('"string1"') |
|---|---|
| 0 | 1 |

7. **JSON_INSERT:**

```
79    -- JSON_INSERT(): Inserting a new key-value pair into the JSON object
80 •  SELECT
81        json_data,
82        JSON_INSERT(json_data, '$.keynew', 'newval')
83    FROM json_demo_table;
84
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| json_data | JSON_INSERT(json_data, '$.keynew', 'newval') |
|---|---|
| {"key1": "value1", "key2": "value2"} | {"key1": "value1", "key2": "value2", "keynew": ... |
| [3, "string1"] | [3, "string1"] |
| {"key3": "value3", "key4": "value4"} | {"key3": "value3", "key4": "value4", "keynew": ... |
| ["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v... | ["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v... |
| {"key1": ["string3", 6], "key2": "value2"} | {"key1": ["string3", 6], "key2": "value2", "keyne... |

## 8. JSON_REMOVE:

```
86      -- JSON_REMOVE(): Removing an element from an array or object
87      SELECT
88          json_data,
89          JSON_REMOVE(json_data, '$[0]')  -- Remove the first element of the array
90      FROM json_demo_table;
91
92      -- JSON_REPLACE(): Replacing the value of a key in the JSON object
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| json_data | JSON_REMOVE(json_data, '$[0]') |
|---|---|
| {"key1": "value1", "key2": "value2"} | {"key1": "value1", "key2": "value2"} |
| [3, "string1"] | ["string1"] |
| {"key3": "value3", "key4": "value4"} | {"key3": "value3", "key4": "value4"} |
| ["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v... | [3, "{\"key5\":\"value5\",\"key6\":\"value6\"}"] |
| {"key1": ["string3", 6], "key2": "value2"} | {"key1": ["string3", 6], "key2": "value2"} |

## 9. JSON_REPLACE: Replaces existing values in a JSON document.

```
92      -- JSON_REPLACE(): Replacing the value of a key in the JSON object
93      SELECT
94          json_data,
95          JSON_REPLACE(json_data, '$.key1', 'val3')
96      FROM json_demo_table;
97
98
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| json_data | JSON_REPLACE(json_data, '$.key1', 'val3') |
|---|---|
| {"key1": "value1", "key2": "value2"} | {"key1": "val3", "key2": "value2"} |
| [3, "string1"] | [3, "string1"] |
| {"key3": "value3", "key4": "value4"} | {"key3": "value3", "key4": "value4"} |
| ["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v... | ["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v... |
| {"key1": ["string3", 6], "key2": "value2"} | {"key1": "val3", "key2": "value2"} |

**10.JSON_SET:** Adds or updates values in a JSON document.

```
98    -- JSON_SET(): Set object with key 'key1' as 'val3'
99    SELECT
100       json_data,
101       JSON_SET(json_data, '$.key1', 'val3')
102   FROM json_demo_table;
103
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| json_data | JSON_SET(json_data, '$.key1', 'val3') |
|---|---|
| {"key1": "value1", "key2": "value2"} | {"key1": "val3", "key2": "value2"} |
| [3, "string1"] | [3, "string1"] |
| {"key3": "value3", "key4": "value4"} | {"key1": "val3", "key3": "value3", "key4": "valu... |
| ["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v... | ["string2", 3, "{\"key5\":\"value5\",\"key6\":\"v... |
| {"key1": ["string3", 6], "key2": "value2"} | {"key1": "val3", "key2": "value2"} |

# 9. DATABASE DEVELOPMENT

Database development plays a crucial role in managing and leveraging data efficiently, securely, and at scale. It is used to create various applications and is essential for businesses seeking to show the importance of data for decision making and innovation.

The below schema has been created from the Enhanced Entity Diagram, that involves all the tables, and meets all the primary and foreign key constraints.

## 9.1 DESCRIPTION:

| Table Name | Description |
|---|---|
| **Admin** | **Usage:** Represents administrators responsible for managing warehouses, products, shopkeepers, and suppliers.<br><br>**Data attributes:**<br>1. AdminID (Integer, Primary Key, Not Null)<br>2. Username (Varchar (50),Not Null)<br>3. Password (Varchar (50),Not Null)<br>4. Email (Varchar (50),Not Null)<br>5. Role (Enum (Manager, Supervisor),Not Null)<br><br>**Constraints:**<br>1. AdminID must be unique and cannot be null. PRIMARY KEY<br>2. Email must follow a valid email format.<br>3. Role is an Enum (Manager, Supervisor)<br><br>**Relationships :** |

| | |
|---|---|
| | 1. Admin manages multiple Warehouses<br>2. Admin manages multiple Products,<br>3. Admin manages multiple Shopkeepers<br>4. Admin manages multiple Suppliers. |
| **Product** | **Usage:** Represents items that are stored in warehouses, supplied by suppliers, and categorized into various product categories.<br><br>**Data Attributes:**<br>1. ProductID (Integer, Primary Key, Not Null)<br>2. ProductName (Varchar (50), Not Null)<br>3. Size (Varchar(50), Not Null)<br>4. PricePerunit (Decimal(10,2),, Not Null)<br>5. QuantityInStock (Integer, Not Null)<br>6. CategoryID (Integer, Foreign Key, Not Null)<br>7. SupplierID (Integer, Foreign Key, Not Null)<br>8. AdminID(Integer, Foreign Key, Not Null)<br><br>**Constraints:**<br>1. ProductID must be unique and cannot be null. PRIMARY KEY<br>2. CategoryID references the Category table. Foreign Key,<br>3. SupplierID references the Supplier table. Foreign Key, |

| | |
|---|---|
| | 4. AdminID references the admin table. Foreign Key<br><br>**Relationships:**<br>1. Product can be involved in multiple BorrowRequests .<br>2. Product Belongs to one Category.<br>3. Product Supplied by one Supplier.<br>4. Each product has an admin |

| | |
|---|---|
| **Warehouse** | **Usage:** Represents storage locations for products and handles stock audits and shipments.<br><br>**Data Attributes:**<br>1. WarehouseID (Integer, Primary Key, Not Null)<br>2. LocationID (Integer, Foreign Key, Not Null)<br>3. MaxCapacity (Integer,Not Null)<br>4. AvailableSpace(Integer,Not Null)<br>5. CurrentCapacity (Integer,Not Null)<br>6. AdminID (references the admin table. Foreign Key, Not Null)<br><br>**Constraints :**<br>1. WarehouseID must be unique and cannot be null.<br>2. LocationID references the Location table.<br><br>**Relationships:**<br>1. Warehouse Stores multiple Storage.<br>2. Warehouse Processes multiple Shipments.<br>3. Warehouse Undergoes one StockAudit . |

| | |
|---|---|
| **Supplier** | **Usage:** Represents entities that supply products to the warehouses.<br><br>**Data Attributes:**<br>1. SupplierID (Integer, Primary Key, Not Null)<br>2. SupplierName ( Varchar (100), Not Null)<br>3. ContactPerson ( Varchar (100), Not Null)<br>4. Email (Varchar (100), Not Null)<br>5. SuppliedCloths (Varchar (100), Not Null)<br>6. AdminID (references the admin table. Foreign Key, Not Null)<br><br>**Constraints:**<br>1. SupplierID must be unique and cannot be null.<br>2. AdminID references the admin table. Foreign Key<br><br>**Relationships:**<br>1. Supplier supplies multiple Products .<br>2. Each Supplier has a admin |

| | |
|---|---|
| **StockAudit** | **Usage:** Represents audits performed on the stock of a warehouse.<br><br>**Data Attributes:**<br>1. AuditID (Integer, Primary Key, Not Null)<br>2. WarehouseID (Integer, Foreign Key, Not Null)<br>3. AuditDate (Date, Not Null)<br>4. AuditorName (Varchar(50),Not Null)<br>5. Discrepancies TEXT<br><br>**Constraints:**<br>1. StockAuditID must be unique and cannot be null.<br>2. WarehouseID references the Warehouse table.<br><br>**Relationships:**<br>1. StockAuditID Performed on one Warehouse. |

| | |
|---|---|
| **Location** | **Usage:** Represents physical locations of warehouses or shops.<br><br>**Data Attributes:**<br>2. LocationID (Integer, Primary Key, Not Null)<br>3. Zipcode( Varchar(55),Not Null)<br>4. Email (Varchar (100),Not Null)<br>5. Street (Varchar (55),Not Null)<br>6. State (Varchar (55),Not Null)<br>7. Country (Varchar (55)Not Null)<br>8. OpeningHours( Varchar(55),)<br>9. ManagerName( Varchar(55),)<br><br>**Constraints:**<br>1. LocationID must be unique and cannot be null.<br><br>**Relationships:**<br>1. Location Contains multiple Warehouses . |
| **Storage** | **Usage :** Represents the many-to-many relationship between Products and Warehouses, indicating which products are stored in which warehouses.<br><br>**Data Attributes**:<br>1. ProductID (Integer, Foreign Key, Not Null)<br>2. WarehouseID (Integer, Foreign Key, Not Null)<br>3. CarbonfootPrint (Double(10,5))<br><br>**Constraints**:<br>1. ProductID references the Product table and cannot be null.<br>2. WarehouseID references the Warehouse table and cannot be null.<br>3. Together, ProductID and WarehouseID form a composite primary key. |

| | |
|---|---|
| | **Relationships**:<br>1. Storage References for one Product.<br>2. Storage References one Warehouse. |
| **Category** | **Usage:** Represents the category of products that are stored in warehouses and supplied by suppliers.<br><br>**Data Attributes:**<br>1. CategoryID (Integer, Primary Key, Not Null)<br>2. CategoryName (Varchar(255), Not Null)<br>3. Description (Text, Nullable)<br>4. CreatedDate (Date, Not Null)<br>5. Seasonality (Varchar(50), Nullable)<br><br>**Constraints:**<br>1. CategoryID must be unique and cannot be null. PRIMARY KEY.<br><br>**Relationships:**<br>1. CategoryID is referenced by the Product table.<br>2. Each Category can have multiple products. |

| | |
|---|---|
| **BorrowRequest** | **Usage:**<br>Represents the request made by shopkeepers to borrow products from warehouses.<br><br>**Data Attributes:**<br>1. RequestID (Integer, Primary Key, Not Null)<br>2. RequestDate (Date, Not Null)<br>3. BorrowStatus (Varchar(50), Not Null)<br>4. ProductID (Integer, Foreign Key, Not Null)<br>5. ShopkeeperID (Integer, Foreign Key, Not Null)<br><br>**Constraints:**<br>1. RequestID must be unique and cannot be null. PRIMARY KEY.<br>2. ProductID references the Product table.<br>3. ShopkeeperID references the Shopkeeper table.<br><br>**Relationships:**<br>1. Each BorrowRequest is associated with one Product.<br>2. Each BorrowRequest is associated with one Shopkeeper. |
| **Transaction** | **Usage:** Represents financial transactions related to orders placed by shopkeepers.<br><br>**Data Attributes:**<br>1. TransactionId (Integer, Primary Key, Not Null)<br>2. TransactionDate (Date, Nullable)<br>3. PaymentMethod (Varchar(45), Nullable)<br>4. TotalAmount (Decimal(10,2), Nullable)<br>5. OrderId (Integer, Foreign Key, Nullable)<br>6. Invoice (Long BLOB, Nullable)<br><br>**Constraints:**<br>1. TransactionId must be unique and cannot be null. PRIMARY KEY.<br>2. OrderId references the Order table.<br><br>**Relationships:** |

| | |
|---|---|
| | 1. Each Transaction is associated with one Order. |

| | |
|---|---|
| **Order** | **Usage**: Represents orders placed by shopkeepers for products from warehouses.<br><br>**Data Attributes:**<br>1. OrderId (Integer, Primary Key, Not Null)<br>2. OrderDate (Date, Not Null)<br>3. Quantity (Integer, Not Null)<br>4. OrderStatus (Varchar(50), Not Null)<br>5. Amount (Decimal(10,2), Not Null)<br>6. ShopkeeperId (Integer, Foreign Key, Not Null)<br><br>**Constraints:**<br>1. OrderId must be unique and cannot be null. PRIMARY KEY.<br>2. ShopkeeperId references the Shopkeeper table.<br><br>**Relationships:**<br>1. Each Order is associated with one Shopkeeper.<br>2. Each Order can have multiple Shipments. |

| | |
|---|---|
| **Shipment** | **Usage:** Represents shipments for the orders placed by shopkeepers, tracking the shipment process.<br><br>**Data Attributes:**<br>1. ShipmentId (Integer, Primary Key, Not Null)<br>2. ShipmentDate (Date, Not Null)<br>3. ShipmentStatus (Varchar(45), Not Null)<br>4. TrackingNumber (Varchar(45), Not Null)<br>5. OrderId (Integer, Foreign Key, Not Null)<br>6. WAREHOUSE_WarehouseID (Integer, Foreign Key, Not Null)<br><br>**Constraints:**<br>1. ShipmentId must be unique and cannot be null. PRIMARY KEY.<br>2. OrderId references the Order table.<br>3. WAREHOUSE_WarehouseID references the Warehouse table.<br><br>**Relationships:**<br>1. Each Shipment is linked to one Order.<br>2. Each Shipment is processed by one Warehouse. |
| Shopkeeper | **Usage:** Represents shopkeepers who place orders and borrow products from warehouses.<br><br>**Data Attributes:**<br>1. ShopkeeperID (Integer, Primary Key, Not Null)<br>2. Username (Varchar(100), Not Null)<br>3. Password (Varchar(100), Not Null)<br>4. FullName (Varchar(100))<br>5. Email (Varchar(100), Not Null)<br>6. ADMIN_AdminID (Integer, Foreign Key, Not Null)<br><br>**Constraints:**<br>1. ShopkeeperID must be unique and cannot be null. PRIMARY KEY.<br>2. ADMIN_AdminID references the Admin table.<br><br>**Relationships:**<br>1. Each Shopkeeper is managed by one Admin. |

| | |
|---|---|
| | 2. Each Shopkeeper can place multiple Orders and submit multiple BorrowRequests. |

# 9.2 CREATE STATEMENTS

## 1. CREATE Warehouse_Management_System_DBMS_Project DATABASE:

```
CREATE DATABASE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`;
USE `Warehouse_Management_System_DBMS_Project` ;
```

## 2. CREATE ADMIN TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`ADMIN` (
  `AdminID` INT NOT NULL AUTO_INCREMENT,
  `Username` VARCHAR (50) NOT NULL,
  `Password` VARCHAR(50) NOT NULL,
  `Role` ENUM('Manager', 'Supervisor') NOT NULL,
```

```
  `Email` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`AdminID`));
```

## 3. CREATE SHOPKEEPER TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`SHOPKEEPER` (
  `ShopkeeperID` INT NOT NULL AUTO_INCREMENT,
  `Username` VARCHAR(100) NOT NULL,
  `Password` VARCHAR(100) NOT NULL,
  `FullName` VARCHAR(100) NULL DEFAULT NULL,
  `Email` VARCHAR(100) NOT NULL,
  `ADMIN_AdminID` INT NOT NULL,
  PRIMARY KEY (`ShopkeeperID`),
  CONSTRAINT `fk_SHOPKEEPER_ADMIN1`
    FOREIGN KEY (`ADMIN_AdminID`)
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`ADMIN` (`AdminID`));
```

## 4. CREATE CATEGORY TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`CATEGORY` (
  `CategoryID` INT NOT NULL AUTO_INCREMENT,
  `CategoryName` VARCHAR(255) NOT NULL,
  `Description` TEXT NULL,
  `CreatedDate` DATE NOT NULL,
  `Seasonality` VARCHAR(50) NOT NULL,
```

```
  PRIMARY KEY (`CategoryID`));
```

## 5. CREATE SUPPLIER TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`SUPPLIER` (
  `SupplierID` INT NOT NULL AUTO_INCREMENT,
  `SupplierName` VARCHAR(100) NOT NULL,
  `ContactPerson` VARCHAR(100) NOT NULL,
  `Email` VARCHAR(100) NOT NULL,
  `SuppliedCloths` VARCHAR(100) NOT NULL,
  `ADMIN_AdminID` INT NOT NULL,
  PRIMARY KEY (`SupplierID`),
  CONSTRAINT `fk_SUPPLIER_ADMIN1`
    FOREIGN KEY (`ADMIN_AdminID`)
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`ADMIN` (`AdminID`));
```

## 6. CREATE LOCATION TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`LOCATION` (
  `locationId` INT NOT NULL AUTO_INCREMENT,
  `ZipCode` VARCHAR(50) NOT NULL,
  `Email` VARCHAR(100) NOT NULL,
  `Street` VARCHAR(50) NOT NULL,
  `ManagerName` VARCHAR(55) NULL,
  `Country` VARCHAR(55) NOT NULL,
```

```
  `OpeningHours` VARCHAR(100) NULL,
  `State` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`locationId`));
```

## 7. CREATE PRODUCT TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`PRODUCT` (
  `productID` INT NOT NULL AUTO_INCREMENT,
  `productName` VARCHAR(100) NOT NULL,
  `CategoryId` INT NOT NULL,
  `Size` VARCHAR(50) NOT NULL,
  `PricePerUnit` DECIMAL(10,2) NOT NULL,
  `QuantityInStock` INT NOT NULL,
  `ADMIN_AdminID` INT NOT NULL,
  `SUPPLIER_SupplierID` INT NOT NULL,
  PRIMARY KEY (`productID`),
  CONSTRAINT ``
    FOREIGN KEY (`CategoryId`)
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`CATEGORY`
(`CategoryID`),
  CONSTRAINT `fk_PRODUCT_ADMIN1`
    FOREIGN KEY (`ADMIN_AdminID`)
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`ADMIN` (`AdminID`),
  CONSTRAINT `fk_PRODUCT_SUPPLIER1`
    FOREIGN KEY (`SUPPLIER_SupplierID`)
```

```
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`SUPPLIER`
(`SupplierID`));
```

## 8. CREATE WAREHOUSE TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`WAREHOUSE` (
  `WarehouseID` INT NOT NULL AUTO_INCREMENT,
  `LocationId` INT NOT NULL,
  `MaxCapacity` INT NOT NULL,
  `CurrentCapacity` INT NOT NULL,
  `AvailableSpace` INT NOT NULL,
  `ADMIN_AdminID` INT NOT NULL,
  PRIMARY KEY (`WarehouseID`),
  CONSTRAINT ``
    FOREIGN KEY (`LocationId`)
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`LOCATION`
(`locationId`),
  CONSTRAINT `fk_WAREHOUSE_ADMIN1`
    FOREIGN KEY (`ADMIN_AdminID`)
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`ADMIN` (`AdminID`));
```

## 9. CREATE BORROWREQUEST TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`BORROWREQUEST` (
```

```
  `RequestID` INT NOT NULL AUTO_INCREMENT,
  `RequestDate` DATE NOT NULL,
  `BorrowStatus` VARCHAR(50) NOT NULL,
  `ProductID` INT NOT NULL,
  `ShopkeeperID` INT NOT NULL,
  PRIMARY KEY (`RequestID`),
  CONSTRAINT ``
    FOREIGN KEY (`ProductID`)
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`PRODUCT`
(`productID`),
  CONSTRAINT ``
    FOREIGN KEY (`ShopkeeperID`)
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`SHOPKEEPER`
(`ShopkeeperID`));
```

## 10. CREATE STOCKAUDIT TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`STOCKAUDIT` (
  `AuditID` INT NOT NULL AUTO_INCREMENT,
  `WarehouseID` INT NOT NULL,
  `AuditDate` DATE NOT NULL,
  `AuditorName` VARCHAR(50) NOT NULL,
  `Discrepancies` TEXT NULL,
  PRIMARY KEY (`AuditID`, `WarehouseID`),
  CONSTRAINT `WarehouseID`
    FOREIGN KEY (`WarehouseID`)
```

```
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`WAREHOUSE`
(`WarehouseID`));
```

## 11. CREATE ORDER TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`ORDER` (
  `OrderId` INT NOT NULL AUTO_INCREMENT,
  `OrderDate` DATE NOT NULL,
  `Quantity` INT NOT NULL,
  `OrderStatus` VARCHAR(50) NOT NULL,
  `Amount` DECIMAL(10,2) NOT NULL,
  `ShopkeeperId` INT NOT NULL,
  PRIMARY KEY (`OrderId`),
  CONSTRAINT `ShopkeeperId`
    FOREIGN KEY (`ShopkeeperId`)
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`SHOPKEEPER`
(`ShopkeeperID`));
```

## 12. CREATE SHIPMENT TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`SHIPMENT` (
  `ShipmentId` INT NOT NULL AUTO_INCREMENT,
  `ShipmentDate` DATE NOT NULL,
  `ShipmentStatus` VARCHAR(45) NOT NULL,
  `TrackingNumber` VARCHAR(45) NOT NULL,
```

```
  `OrderId` INT NOT NULL,
  `WAREHOUSE_WarehouseID` INT NOT NULL,
  PRIMARY KEY (`ShipmentId`),
  CONSTRAINT `OrderId`
    FOREIGN KEY (`OrderId`)
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`ORDER` (`OrderId`),
  CONSTRAINT `fk_Shipment_WAREHOUSE1`
    FOREIGN KEY (`WAREHOUSE_WarehouseID`)
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`WAREHOUSE`
(`WarehouseID`));
```

.

## 13. CREATE STORAGE TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`STORAGE` (
  `PRODUCT_productID` INT NOT NULL AUTO_INCREMENT,
  `WAREHOUSE_WarehouseID` INT NOT NULL,
  `CarbonFootprint` DECIMAL(10,5) NULL,
  PRIMARY KEY (`PRODUCT_productID`, `WAREHOUSE_WarehouseID`),
  CONSTRAINT `fk_PRODUCT_has_WAREHOUSE_PRODUCT1`
    FOREIGN KEY (`PRODUCT_productID`)
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`PRODUCT`
(`productID`),
  CONSTRAINT `fk_PRODUCT_has_WAREHOUSE_WAREHOUSE1`
```

```
    FOREIGN KEY (`WAREHOUSE_WarehouseID`)
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`WAREHOUSE`
(`WarehouseID`));
```

## 14. CREATE TRANSACTION TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`TRANSACTION` (
  `TransactionId` INT NOT NULL AUTO_INCREMENT,
  `TransactionDate` DATE NOT NULL,
  `PaymentMethod` VARCHAR(45) NOT NULL,
  `TotalAmount` DECIMAL(10,2) NOT NULL,
  `OrderId` INT NOT NULL,
  `Invoice` LONGBLOB,
  PRIMARY KEY (`TransactionId`),
  CONSTRAINT `fk_Transaction_OrderId`
    FOREIGN KEY (`OrderId`)
    REFERENCES
`Warehouse_Management_System_DBMS_Project`.`ORDER` (`OrderId`));
```

# 10.LOADING DATA AND PERFORMANCE ENHANCEMENT

## 10.1 HANDLING KEY CONSTRAINTS:

**Case 1: Foreign Key Constraint**

a) Insert parent ADMIN records first

```
insert into ADMIN(Username,Password,Role,email)
value (
'Sumanth','password123',1,'sumanth@gmail.com'
);
```

b) Then insert child SHOPKEEPER records

```
insert into
SHOPKEEPER(Username,Password,FullName,Email,ADMIN_Admin
Id)
value (
'AbhijeetC','password123','Abhijeet',abhi@gmail.com',1
);
```

c) Insert second value in shopkeeper

```
insert into
SHOPKEEPER(Username,Password,FullName,Email,ADMIN_Admin
ID)
value (
Sagar,'pass123’,‘SagarS,sagar@gmail.com',2
);
```

d) Error Occurred

**Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails**

```
(`warehouse_management_system_dbms_project`.`shopkeeper
`, CONSTRAINT `fk_SHOPKEEPER_ADMIN1` FOREIGN KEY
(`ADMIN_AdminID`) REFERENCES `admin` (`AdminID`))
```

e) Disable FK checks

```
SET foreign_key_checks = 0;
```

f) Insert SHOPKEEPER records with values without foreign key issue

```
insert into
SHOPKEEPER(Username,Password,FullName,Email,ADMIN_Admin
ID)
value (
Sagar,'pass123',SagarS,sagar@gmail.com',2
);
```

g) Enable FK checks again

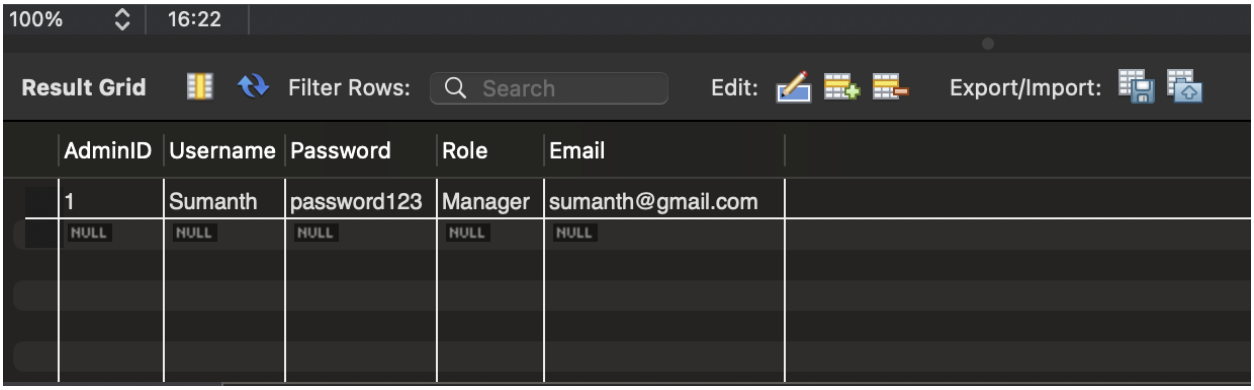
```
SET foreign_key_checks = 1;
```

**Description:**

• The admin table is the parent table of the Shopkeeper table via the ADMIN_AdminID foreign key.
• The admin records are inserted first to ensure the foreign key references exist before inserting child records.
• The Shopkeeper records that reference AdminIDs are then inserted.
• Foreign key checks are disabled to allow inserting a shopkeeper with an invalid foreign key reference.
• Foreign key checks are re-enabled after insertion is complete to restore validation.

**Case 2: Primary Key**

a) Current Shopkeeper records first




b) **Inserting value with same adminID**

```
insert into ADMIN(Username,Password,Role,email)
value (
1,'Abhijeet','password123',1,'abhi@gmail.com'
);
```

c) **Error occurred**

**Error Code: 1136. Column count doesn't match value count at row 1**

**d) Using Auto Increment to change value of inserted data**

```
insert into ADMIN(Username,Password,Role,email)
value (
'Abhijeet','password123',1,'abhi@gmail.com'
);
```

**Description:**

- The `Shopkeeper` table is shown to see current values in it.
- The `Shopkeeper` records are inserted first to ensure the foreign key references exist before inserting dependent records.
- An attempt was made to insert a new `Shopkeeper` record with the same `AdminID`, which led to an error due to column mismatch.
- The error occurred because the column count in the `INSERT` statement did not match the value count.
- The issue was resolved by using the auto-increment feature for `AdminID`, allowing the database to automatically generate a unique `AdminID` for the new record.

**Case 3: Incorrect data type**

a) **Inserting right data into `BORROWREQUEST table`**

```
    insert into
BORROWREQUEST(RequestDate,BorrowStatus,ProductID,Shopke
eperID)
    value (
    '2024-09-30', 'Pending', 1, 1
    );
```

b) **Wrong data into BORROWREQUEST table**

```
    INSERT INTO BORROWREQUEST (RequestDate,
    BorrowStatus, ProductID, ShopkeeperID) VALUES
    ('invalid_date', 12345, 'ABC', 'XYZ');
```

**c) Error occurred**

```
Error Code: 1292. Incorrect date value:
'invalid_date' for column 'RequestDate' at row 1
```

**Description:**

- A correct record is inserted into the `BORROWREQUEST` table with valid data types for `RequestDate`, `BorrowStatus`, `ProductID`, and `ShopkeeperID`.
- An attempt is made to insert a new `BORROWREQUEST` record with invalid data types, leading to an error due to an incorrect date format and mismatched data types in other columns.
- The error occurred because the `RequestDate` value was not in the proper `DATE` format, and invalid data types were used for `ProductID` and `ShopkeeperID`.
- The issue was resolved by ensuring valid `DATE` values and matching the expected data types for each column.

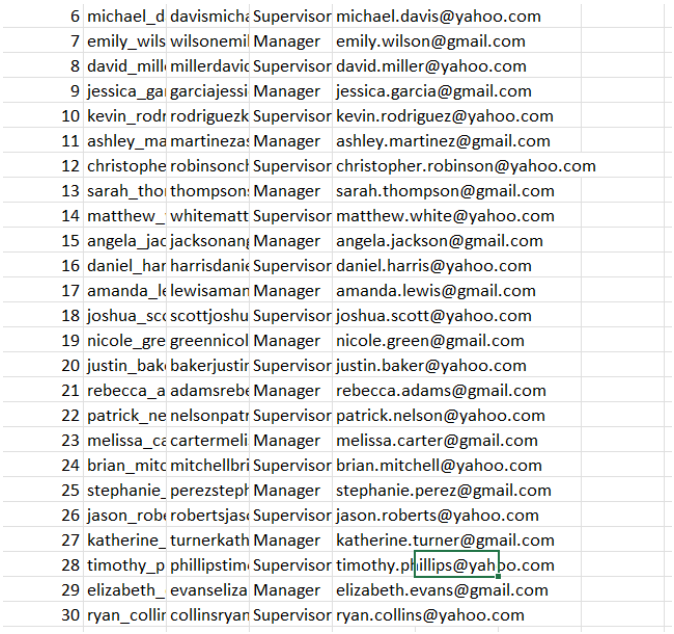
# 10.2 IMPORTING DATA:

Importing sample data into a warehouse management system focused on dress-related products. It includes admin and shopkeeper details, product categories like "Summer Dress", supplier and warehouse information, borrow requests, stock audits, orders, shipments, transactions, and storage details. The data illustrates key system relationships and operations, such as product management, orders, and warehouse capacity.

1. `LOAD DATA INFILE 'admin.csv' INTO TABLE ADMIN`

```
FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n';
```



| | | | | |
|---|---|---|---|---|
| 6 | michael_d | davismich | Supervisor | michael.davis@yahoo.com |
| 7 | emily_wils | wilsonemi | Manager | emily.wilson@gmail.com |
| 8 | david_mill | millerdavic | Supervisor | david.miller@yahoo.com |
| 9 | jessica_ga | garciajessi | Manager | jessica.garcia@gmail.com |
| 10 | kevin_rodr | rodriguezk | Supervisor | kevin.rodriguez@yahoo.com |
| 11 | ashley_ma | martineza | Manager | ashley.martinez@gmail.com |
| 12 | christophe | robinsoncl | Supervisor | christopher.robinson@yahoo.com |
| 13 | sarah_tho | thompson | Manager | sarah.thompson@gmail.com |
| 14 | matthew_ | whitematt | Supervisor | matthew.white@yahoo.com |
| 15 | angela_jac | jacksonan | Manager | angela.jackson@gmail.com |
| 16 | daniel_har | harrisdanie | Supervisor | daniel.harris@yahoo.com |
| 17 | amanda_le | lewisaman | Manager | amanda.lewis@gmail.com |
| 18 | joshua_sc | scottjoshu | Supervisor | joshua.scott@yahoo.com |
| 19 | nicole_gre | greennicol | Manager | nicole.green@gmail.com |
| 20 | justin_bak | bakerjustir | Supervisor | justin.baker@yahoo.com |
| 21 | rebecca_a | adamsrebe | Manager | rebecca.adams@gmail.com |
| 22 | patrick_ne | nelsonpatr | Supervisor | patrick.nelson@yahoo.com |
| 23 | melissa_ca | cartermeli | Manager | melissa.carter@gmail.com |
| 24 | brian_mitc | mitchellbri | Supervisor | brian.mitchell@yahoo.com |
| 25 | stephanie_ | perezsteph | Manager | stephanie.perez@gmail.com |
| 26 | jason_rob | robertsjas | Supervisor | jason.roberts@yahoo.com |
| 27 | katherine_ | turnerkath | Manager | katherine.turner@gmail.com |
| 28 | timothy_p | phillipstim | Supervisor | timothy.phillips@yahoo.com |
| 29 | elizabeth_ | evanseliza | Manager | elizabeth.evans@gmail.com |
| 30 | ryan_collir | collinsryan | Supervisor | ryan.collins@yahoo.com |

2. 
```
LOAD DATA INFILE shopkeeper.csv' INTO TABLE SHOPKEEPER
FIELDS TERMINATED BY ','

 LINES TERMINATED BY '\n';
```

| | | | | | |
|---|---|---|---|---|---|
| 6 | Jessica | shopper30 | Jessica Ga | jessica.gar | 6 |
| 7 | Kevin | shopper40 | Kevin Rodr | kevin.rodri | 7 |
| 8 | Ashley | shopper50 | Ashley Ma | ashley.ma | 8 |
| 9 | Christophe | shopper60 | Christophe | christophe | 9 |
| 10 | Sarah | shopper70 | Sarah Thor | sarah.thor | 10 |
| 11 | Matthew | shopper80 | Matthew \ | matthew.v | 11 |
| 12 | Angela | shopper90 | Angela Jac | angela.jacl | 12 |
| 13 | Daniel | shopper10 | Daniel Har | daniel.har | 13 |
| 14 | Amanda | shopper12 | Amanda Le | amanda.le | 14 |
| 15 | Joshua | shopper13 | Joshua Scc | joshua.sco | 15 |
| 16 | Nicole | shopper14 | Nicole Gre | nicole.gre | 16 |
| 17 | Justin | shopper15 | Justin Bake | justin.bake | 17 |
| 18 | Rebecca | shopper16 | Rebecca A | rebecca.ac | 18 |
| 19 | Patrick | shopper17 | Patrick Ne | patrick.nel | 19 |
| 20 | Melissa | shopper18 | Melissa Ca | melissa.ca | 20 |
| 21 | Brian | shopper19 | Brian Mitc | brian.mitc | 21 |
| 22 | Stephanie | shopper20 | Stephanie | stephanie. | 22 |
| 23 | Jason | shopper21 | Jason Rob | jason.robe | 23 |
| 24 | Katherine | shopper22 | Katherine | katherine. | 24 |
| 25 | Timothy | shopper23 | Timothy Pl | timothy.ph | 25 |
| 26 | Elizabeth | shopper24 | Elizabeth E | elizabeth.e | 26 |
| 27 | Ryan | shopper25 | Ryan Collir | ryan.collin | 27 |
| 28 | Lauren | shopper26 | Lauren Par | lauren.par | 28 |
| 29 | William | shopper27 | William Hu | william.hu | 29 |

Similarly loaded sample data for other tables.

3.
```
LOAD DATA INFILE category.csv' INTO TABLE CATEGORY
FIELDS TERMINATED BY ','
 LINES TERMINATED BY '\n';
```

4.
```
LOAD DATA INFILE supplier.csv' INTO TABLE SUPPLIER
FIELDS TERMINATED BY ','
 LINES TERMINATED BY '\n';
```

**Description**:

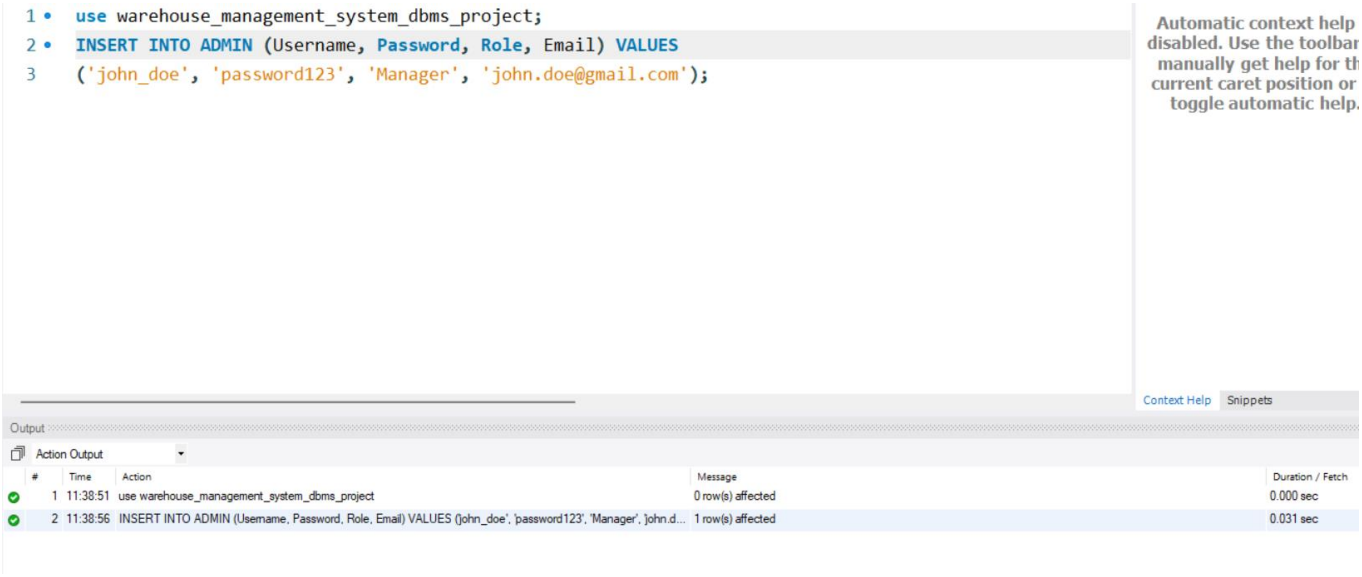
- Use LOAD DATA LOCAL INFILE to import CSV file.
- Specify CSV file path in 'filename'.
- Set the field separator, enclosure, and line terminator correctly. • Use IGNORE 1 ROWS to skip header row.
- Match the columns in the CSV to the table columns
- Correct **ADMIN** records are inserted, assigning unique usernames, roles, and emails for system administrators.

- The **SHOPKEEPER** table is populated with valid shopkeeper details, linking each shopkeeper to an admin via a foreign key.
- **CATEGORY** table entries represent different product types, such as dresses and accessories, along with details like seasonality and description.
- Supplier information is added in the **SUPPLIER** table, ensuring each supplier is associated with an admin for reference.
- The **LOCATION** table records warehouse locations, including address, manager name, and operational hours.
- **PRODUCT** entries include dress-related items such as "Summer Dress" and "Leather Belt," with details on pricing, stock, and size.
- **WAREHOUSE** data reflects the warehouse capacities, linking them to their respective locations and administrators.
- Sample **BORROWREQUEST** entries reflect shopkeepers borrowing products, with pending and approved statuses.
- **STOCKAUDIT** entries document warehouse audits, including discrepancies such as missing stock.
- **ORDER** records show shopkeeper orders, their statuses, quantities, and total amounts for different products.
- **SHIPMENT** data includes delivery statuses and tracking information, linking each shipment to a specific order and warehouse.
- **TRANSACTION** entries track payments made via different methods for completed orders.
- The **STORAGE** table records product storage details, including the carbon footprint of each product within the warehouses.

# 10.3 OPTIMIZATION:

## 1. INSERTION OPTIMIZATION:

Using single insert statements as shown below will insert the values into the table however, it will take more amount of time to execute each statement.[17]




To fix this we Used bulk INSERT statements instead of INSERTING records individually:


Hence the execution time to insert each row with a single insert statement would be 5*0.031= 0.155 secs approximately, whereas to execute the bulk statement to insert values, it only took 0.016 secs as shown above.

Disabled indexes and foreign key checks before insertion:

SET FOREIGN_KEY_CHECKS = 0;

**INSERT STATEMENTS:**

Used multiple VALUES lists in one INSERT statement:

**ADMIN insert statements:**

```
INSERT INTO ADMIN (Username, Password, Role, Email)
VALUES
('john_doe', 'password123', 'Manager',
'john.doe@gmail.com'),
('jane_smith', 'adminpass', 'Supervisor',
'jane.smith@gmail.com'),
('alice_lee', 'leealice456', 'Manager',
'alice.lee@gmail.com'),
```

```
('bob_jones', 'bobsecurepass', 'Supervisor',
'bob.jones@gmail.com'),

('susan_clark', 'clarksecure123', 'Manager',
'susan.clark@gmail.com');
```

**SHOPKEEPER insert statements:**

```
INSERT INTO SHOPKEEPER (Username, Password, FullName,
Email, ADMIN_AdminID) VALUES

('David', 'shopper123', 'David Johnson',
'david.johnson@gmail.com', 1),

('Emily', 'shopper456', 'Emily Davis',
'emily.davis@gmail.com', 2),

('Brown', 'shopper789', 'Michael Brown',
'michael.brown@gmail.com', 3),

('Sarah', 'shopper101', 'Sarah Wilson',
'sarah.wilson@gmail.com', 4),

('James', 'shopper202', 'James Miller',
'james.miller@gmail.com', 5);
```

**CATEGORY insert statements:**

```
INSERT INTO CATEGORY (CategoryName, Description,
CreatedDate, Seasonality) VALUES

('Summer Collection', 'Light and breezy fabrics perfect
for summer.', '2023-06-01', 'Summer'),

('Winter Wear', 'Heavy and warm clothing for winter.',
'2023-10-01', 'Winter'),

('Spring Collection', 'Floral and light spring
designs.', '2023-03-01', 'Spring'),
```

```
('Fall Collection', 'Warm autumn colors and designs.',
'2023-09-01', 'Fall'),

('Accessories', 'Various fashion accessories for all
seasons.', '2023-05-15', 'All Seasons');
```

**SUPPLIER insert statements:**

```
INSERT INTO SUPPLIER (SupplierName, ContactPerson,
Email, SuppliedCloths, ADMIN_AdminID) VALUES

('Fabric World', 'George Thomas',
'george.thomas@gmail.com', 'Cotton, Silk', 1),

('Textile Co', 'Laura Anderson',
'laura.anderson@gmail.com', 'Wool, Denim', 2),

('Cloth Suppliers Inc', 'Kevin White',
'kevin.white@gmail.com', 'Linen, Polyester', 3),

('Global Fabrics', 'Angela Green',
'angela.green@gmail.com', 'Leather, Velvet', 4),

('Elite Clothings', 'Matthew Hall',
'matthew.hall@gmail.com', 'Satin, Silk', 5);
```

**LOCATION insert statements:**

```
INSERT INTO LOCATION (ZipCode, Email, Street,
ManagerName, Country, OpeningHours, State) VALUES

('10001', 'john@warehouse.com', '5th Ave', 'John Snow',
'USA', '9 AM - 5 PM', 'NY'),

('20002', 'lannister@warehouse.com', 'Broadway St',
'Cersei Lannister', 'USA', '10 AM - 6 PM', 'CA'),
```

```
('30003', 'targaryen@warehouse.com', 'Elm St',
'Daenerys Targaryen', 'USA', '8 AM - 4 PM', 'TX'),

('40004', 'stark@warehouse.com', 'Main St', 'Arya
Stark', 'USA', '7 AM - 3 PM', 'FL'),

('50005', 'jeff@warehouse.com', 'Sunset Blvd', 'Jeff
Bezos', 'USA', '11 AM - 7 PM', 'NV');
```

**PRODUCT insert statements:**

```
INSERT INTO PRODUCT (productName, CategoryId, Size,
PricePerUnit, QuantityInStock, ADMIN_AdminID,
SUPPLIER_SupplierID) VALUES

('Cotton Shirt', 1, 'L', 19.99, 50, 1, 1),

('Woolen Jacket', 2, 'M', 49.99, 30, 2, 2),

('Silk Scarf', 5, 'One Size', 14.99, 100, 3, 3),

('Denim Jeans', 1, 'XL', 39.99, 40, 4, 4),

('Leather Gloves', 2, 'M', 24.99, 20, 5, 5);
```

**WAREHOUSE insert statements:**

```
INSERT INTO WAREHOUSE (LocationId, MaxCapacity,
CurrentCapacity, AvailableSpace, ADMIN_AdminID) VALUES

(1, 1000, 800, 200, 1),

(2, 1500, 1000, 500, 2),

(3, 2000, 1800, 200, 3),

(4, 1200, 900, 300, 4),

(5, 1700, 1500, 200, 5);
```

**BORROWREQUEST insert statements:**

```
INSERT INTO BORROWREQUEST (RequestDate, BorrowStatus,
ProductID, ShopkeeperID) VALUES
('2024-08-01', 'Pending', 1, 1),
('2024-08-02', 'Approved', 2, 2),
('2024-08-03', 'Rejected', 3, 3),
('2024-08-04', 'Pending', 4, 4),
('2024-08-05', 'Approved', 5, 5);
```

**STOCKAUDIT insert statements:**

```
INSERT INTO STOCKAUDIT (WarehouseID, AuditDate,
AuditorName, Discrepancies) VALUES
(1, '2024-09-01', 'Tom Hardy', 'No discrepancies'),
(2, '2024-09-02', 'Emma Watson', 'Minor stock issues'),
(3, '2024-09-03', 'Chris Evans', 'Overstock in section
B'),
(4, '2024-09-04', 'Scarlett Johansson', 'Missing items
in section D'),
(5, '2024-09-05', 'Robert Downey Jr', 'Stock mismatch
in section A');
```

**ORDER insert statements:**

```
INSERT INTO
Warehouse_Management_System_DBMS_Project.order
```

```
(OrderDate, Quantity, OrderStatus, Amount,
ShopkeeperId) VALUES
('2024-09-01', 10, 'Shipped', 199.90, 1),
('2024-09-02', 5, 'Delivered', 99.95, 2),
('2024-09-03', 20, 'Processing', 399.80, 3),
('2024-09-04', 15, 'Canceled', 299.85, 4),
('2024-09-05', 8, 'Pending', 159.92, 5);
```

**SHIPMENT insert statements:**

```
INSERT INTO SHIPMENT (ShipmentDate, ShipmentStatus,
TrackingNumber, OrderId, WAREHOUSE_WarehouseID) VALUES
('2024-09-06', 'Shipped', 'TRK12345', 1, 1),
('2024-09-07', 'Delivered', 'TRK12346', 2, 2),
('2024-09-08', 'In Transit', 'TRK12347', 3, 3),
('2024-09-09', 'Processing', 'TRK12348', 4, 4),
('2024-09-10', 'Pending', 'TRK12349', 5, 5);
```

**TRANSACTION insert statements:**

```
INSERT INTO TRANSACTION (TransactionDate,
PaymentMethod, TotalAmount, OrderId, Invoice) VALUES
('2024-09-06', 'Credit Card', 199.90, 1, NULL),
('2024-09-07', 'PayPal', 99.95, 2, NULL),
('2024-09-08', 'Bank Transfer', 399.80, 3, NULL),
('2024-09-09', 'Debit Card', 299.85, 4, NULL),
('2024-09-10', 'Credit Card', 159.92, 5, NULL);
```

**STORAGE insert statements:**

```
INSERT INTO STORAGE (PRODUCT_productID,
WAREHOUSE_WarehouseID, CarbonFootprint) VALUES
(1, 1, 2.34567),
(2, 2, 1.98765),
(3, 3, 3.45678),
(4, 4, 1.23456),
(5, 5, 2.87654);
```

## 10.4 INDEX OPTIMIZATION

The best way to improve the performance of select operations is to create indexes on one or more of the columns that are tested in the query. The index entries act like pointers to the table rows, allowing the query to quickly determine which rows match a condition in the WHERE clause, and retrieve the other column values for those rows.[15][16]

**INDEX ON ADMIN:**

```
CREATE INDEX idx_admin_username ON ADMIN(Username);
CREATE INDEX idx_admin_email ON ADMIN(Email);
```

**INDEX ON SHOPKEEPER:**

```
CREATE INDEX idx_shopkeeper_username ON
SHOPKEEPER(Username);

CREATE INDEX idx_shopkeeper_email ON SHOPKEEPER(Email);

CREATE INDEX idx_shopkeeper_admin_id ON
SHOPKEEPER(ADMIN_AdminID);
```

**INDEX ON CATEGORY:**

```
CREATE INDEX idx_category_name ON
CATEGORY(CategoryName);
```

**INDEX ON SUPPLIER:**

```
CREATE INDEX idx_supplier_name ON
SUPPLIER(SupplierName);

CREATE INDEX idx_supplier_admin_id ON
SUPPLIER(ADMIN_AdminID);
```

**INDEX ON LOCATION:**

```
CREATE INDEX idx_location_zipcode ON LOCATION(ZipCode);

CREATE INDEX idx_location_country ON LOCATION(Country);

CREATE INDEX idx_location_state ON LOCATION(State);
```

**INDEX ON PRODUCT:**

```
CREATE INDEX idx_product_name ON PRODUCT(productName);

CREATE INDEX idx_product_category ON
PRODUCT(CategoryId);
```

```
CREATE INDEX idx_product_admin_id ON
PRODUCT(ADMIN_AdminID);

CREATE INDEX idx_product_supplier_id ON
PRODUCT(SUPPLIER_SupplierID);
```

**INDEX ON WAREHOUSE:**

```
CREATE INDEX idx_warehouse_location ON
WAREHOUSE(LocationId);

CREATE INDEX idx_warehouse_admin_id ON
WAREHOUSE(ADMIN_AdminID);
```

**INDEX ON BORROWREQUEST:**

```
CREATE INDEX idx_borrowrequest_date ON
BORROWREQUEST(RequestDate);

CREATE INDEX idx_borrowrequest_product ON
BORROWREQUEST(ProductID);

CREATE INDEX idx_borrowrequest_shopkeeper ON
BORROWREQUEST(ShopkeeperID);
```

**INDEX ON ORDER:**

```
CREATE INDEX idx_order_date ON `ORDER`(OrderDate);

CREATE INDEX idx_order_status ON `ORDER`(OrderStatus);

CREATE INDEX idx_order_shopkeeper ON
`ORDER`(ShopkeeperId);
```

**INDEX ON SHIPMENT:**

```
CREATE INDEX idx_shipment_date ON
SHIPMENT(ShipmentDate);

CREATE INDEX idx_shipment_status ON
SHIPMENT(ShipmentStatus);

CREATE INDEX idx_shipment_order ON SHIPMENT(OrderId);

CREATE INDEX idx_shipment_warehouse ON
SHIPMENT(WAREHOUSE_WarehouseID);
```

**INDEX ON TRANSACTION:**

```
CREATE INDEX idx_transaction_date ON
`TRANSACTION`(TransactionDate);

CREATE INDEX idx_transaction_order ON
`TRANSACTION`(OrderId);
```

**To test the performance difference we have executed below join query before and after creating index.**

**Query:**

```
SELECT s.ShopkeeperID, s.Username,  p.productName,
c.CategoryName

FROM SHOPKEEPER s JOIN

    PRODUCT p ON s.ShopkeeperID = p.ADMIN_AdminID

JOIN

    CATEGORY c ON p.CategoryId = c.CategoryID

WHERE

    s.Username = 'DAVID';
```

**EXPLAIN Plan before INDEX Creation.**

```
EXPLAIN: -> Nested loop inner join  (cost=5.18 rows=2.9) (actual time=0.109..0.109 rows=0 loops=1)
           -> Nested loop inner join  (cost=4.17 rows=2.9) (actual time=0.108..0.108 rows=0 loops=1)
              -> Filter: (s.Username = 'shopkeeper1')  (cost=3.15 rows=2.9) (actual time=0.107..0.107 rows=0 loops=1)
                 -> Table scan on s  (cost=3.15 rows=29) (actual time=0.0869..0.099 rows=29 loops=1)
```

**EXPLAIN Plan after INDEX Creation.**

```
EXPLAIN: -> Nested loop inner join  (cost=1.05 rows=1) (actual time=0.0208..0.0208 rows=0 loops=1)
           -> Nested loop inner join  (cost=0.7 rows=1) (actual time=0.0204..0.0204 rows=0 loops=1)
              -> Covering index lookup on s using idx_shopkeeper_username (Username='shopkeeper1')  (cost=0.35 rows=1) (actual time=0.0195..0.0195 rows=0 loops=1)
              -> Index lookup on p using idx_product_admin_id (ADMIN_AdminID=s.ShopkeeperID)  (cost=0.35 rows=1) (never executed)
```

# 10.5 NORMALIZATION CHECK

Normalization organizes the columns and tables of a database to ensure that database integrity constraints properly execute their dependencies. It is a systematic technique of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update, and Deletion anomalies.

1. **First Normal Form(1NF):**

   The first normal form states that each table cell should contain a single value and each record needs to be unique.

- For example, in the Admin table, all the values are unique and each cell contains only a single value.

```
5 •  SELECT * FROM ADMIN;
```

Result Grid | Filter Rows: | Edit: | Export/Import

| AdminID | Username | Password | Role | Email |
|---|---|---|---|---|
| 1 | john_doe | password123 | Manager | john.doe@gmail.com |
| 2 | jane_smith | adminpass | Supervisor | jane.smith@gmail.com |
| 3 | alice_lee | leealice456 | Manager | alice.lee@gmail.com |
| 4 | bob_jones | bobsecurepass | Supervisor | bob.jones@gmail.com |

- For example, in the Product table, all the values are unique and each cell contains only a single value.

```
5 •  SELECT * FROM product;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| productID | productName | CategoryId | Size | PricePerUnit | QuantityInStock | ADMIN_AdminID | SUPPLIER_SupplierID |
|---|---|---|---|---|---|---|---|
| 1 | Cotton Shirt | 1 | L | 19.99 | 50 | 1 | 1 |
| 2 | Woolen Jacket | 2 | M | 49.99 | 30 | 2 | 2 |
| 3 | Silk Scarf | 5 | One Size | 14.99 | 100 | 3 | 3 |
| 4 | Denim Jeans | 1 | XL | 39.99 | 40 | 4 | 4 |
| 5 | Leather Gloves | 2 | M | 24.99 | 20 | 5 | 5 |

## 2. Second Normal Form(2NF):

The second normal form states that it should be in INF and it does not have any non-prime attribute that is functionally dependent on any proper subset of any candidate key of the relation.

- For example, in the below tables Order, Supplier, and Shopkeeper, it is clearly shown that they are in 1 NF and it does not contain any partial dependency, i.e., all non-prime attributes are fully functionally dependent on the primary key.

```
5 •  SELECT * FROM `ORDER`;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

| OrderId | OrderDate | Quantity | OrderStatus | Amount | ShopkeeperId |
|---|---|---|---|---|---|
| 1 | 2024-09-01 | 10 | Shipped | 199.90 | 1 |
| 2 | 2024-09-02 | 5 | Delivered | 99.95 | 2 |
| 3 | 2024-09-03 | 20 | Processing | 399.80 | 3 |
| 4 | 2024-09-04 | 15 | Canceled | 299.85 | 4 |
| 5 | 2024-09-05 | 8 | Pending | 159.92 | 5 |

```
6 •  SELECT * FROM Supplier;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| SupplierID | SupplierName | ContactPerson | Email | SuppliedCloths | ADMIN_AdminID |
|---|---|---|---|---|---|
| 1 | Fabric World | George Thomas | george.thomas@gmail.com | Cotton, Silk | 1 |
| 2 | Textile Co | Laura Anderson | laura.anderson@gmail.com | Wool, Denim | 2 |
| 3 | Cloth Suppliers Inc | Kevin White | kevin.white@gmail.com | Linen, Polyester | 3 |
| 4 | Global Fabrics | Angela Green | angela.green@gmail.com | Leather, Velvet | 4 |
| 5 | Elite Clothings | Matthew Hall | matthew.hall@gmail.com | Satin, Silk | 5 |

```
6 •  SELECT * FROM Shopkeeper;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| ShopkeeperID | Username | Password | FullName | Email | ADMIN_AdminID |
|---|---|---|---|---|---|
| 1 | David | shopper123 | David Johnson | david.johnson@gmail.com | 1 |
| 2 | Emily | shopper456 | Emily Davis | emily.davis@gmail.com | 2 |
| 3 | Brown | shopper789 | Michael Brown | michael.brown@gmail.com | 3 |
| 4 | Sarah | shopper101 | Sarah Wilson | sarah.wilson@gmail.com | 4 |

### 3. Third Normal Form (3NF):

The third normal form states that it should be in 2NF and has no transitive functional dependencies.

- For example, in the tables Warehouse and BorrowRequest there is no transitive dependency between the attributes. Hence in 3NF.

```
6 •  SELECT * FROM Warehouse;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

| WarehouseID | LocationId | MaxCapacity | CurrentCapacity | AvailableSpace | ADMIN_AdminID |
|---|---|---|---|---|---|
| 1 | 1 | 1000 | 800 | 200 | 1 |
| 2 | 2 | 1500 | 1000 | 500 | 2 |
| 3 | 3 | 2000 | 1800 | 200 | 3 |
| 4 | 4 | 1200 | 900 | 300 | 4 |
| 5 | 5 | 1700 | 1500 | 200 | 5 |

```
6 •  SELECT * FROM BorrowRequest;
```

Result Grid | Filter Rows: | Edit:

| RequestID | RequestDate | BorrowStatus | ProductID | ShopkeeperID |
|---|---|---|---|---|
| 1 | 2024-08-01 | Pending | 1 | 1 |
| 2 | 2024-08-02 | Approved | 2 | 2 |
| 3 | 2024-08-03 | Rejected | 3 | 3 |
| 4 | 2024-08-04 | Pending | 4 | 4 |
| 5 | 2024-08-05 | Approved | 5 | 5 |

- For example, in the tables Location and STOCKAUDIT there exists a transitive dependency between each Zip Code and each stockaudit attrtibutes. Hence, we created a new table called ZIPCODE_INFO to remove the transitive dependency, which has ZipCode as the primary key. In STOCKAUDIT, used AuditID as the primary key to address the partial dependency.

```
3      -- Create ZIPCODE_INFO table
4  •   CREATE TABLE ZIPCODE_INFO (
5          ZipCode VARCHAR(10) PRIMARY KEY,
6          State VARCHAR(50),
7          Country VARCHAR(50)
8      );
9
10     -- Modify LOCATION table
11 •   ALTER TABLE LOCATION
12     DROP COLUMN State,
13     DROP COLUMN Country;
14
15 •   ALTER TABLE LOCATION
16     ADD CONSTRAINT fk_zipcode_info FOREIGN KEY (ZipCode) REFERENCES ZIPCODE_INFO (ZipCode);
```

```
4 •   select * from LOCATION;
5
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| locationId | ZipCode | Email | Street | ManagerName | OpeningHours |
|---|---|---|---|---|---|
| 1 | 10001 | john@warehouse.com | 5th Ave | John Snow | 9 AM - 5 PM |
| 2 | 20002 | lannister@warehouse.com | Broadway St | Cersei Lannister | 10 AM - 6 PM |
| 3 | 30003 | targaryen@warehouse.com | Elm St | Daenerys Targaryen | 8 AM - 4 PM |

```
4 •  select * from ZIPCODE_INFO;
5
```

Result Grid | Filter Rows:

| ZipCode | State | Country |
|---|---|---|
| 12345 | NY | USA |
| NULL | NULL | NULL |

```
4    -- Modify STOCKAUDIT table to use AuditID as the primary key
5 •  ALTER TABLE STOCKAUDIT DROP PRIMARY KEY,ADD PRIMARY KEY (AuditID);
6 •  select * from STOCKAUDIT;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap C

| AuditID | WarehouseID | AuditDate | AuditorName | Discrepancies |
|---|---|---|---|---|
| 1 | 1 | 2024-09-01 | Tom Hardy | No discrepancies |
| 2 | 2 | 2024-09-02 | Emma Watson | Minor stock issues |
| 3 | 3 | 2024-09-03 | Chris Evans | Overstock in section B |
| 4 | 4 | 2024-09-04 | Scarlett Johansson | Missing items in section D |

# 11.APPLICATION DEVELOPMENT

## 11.1 GRAPHICAL USER EXPERIENCE DESIGN

### 11.1.1 GUE ADMIN POV

Warehouse Management System - Admin POV

1. **Start**
    - The system begins here, initiating the admin interaction.
2. **Admin Login**
    - Admin enters their credentials (username and password).
    - If invalid, the system shows an **"Invalid Username/Password"** error.
3. **Authentication**
    - If the login is successful, the admin proceeds to the **Landing Page** (Admin Dashboard).
4. **Landing Page (Admin Dashboard)**

    - The admin can access various tables and features. From here, the admin can navigate to:
        - **Product Table**: To view, add, or edit product information (e.g., dress-related products).
        - **Warehouse Table**: To manage warehouse details (location, capacity, etc.).
        - **Shopkeeper Table**: To update or check current stock levels for each Shopkeeper.
        - **Supplier Table**: To handle supplies orders, stock allocation, and shipment.
5. **Authentication Requirement**
    - Access to these tables and processes is restricted to authenticated admin users. Unauthorized users are redirected back to the login page.
6. **Actions for Admin**
    - After authentication, the admin can perform the following tasks:
        - **Add/Edit Records**: Add or modify information in the product, warehouse, and stock tables.
        - **Delete Records**: Remove outdated or incorrect records from the system.
7. **Logout or End**
    - Admin can choose to logout, returning to the start page, or the process ends when all tasks are complete.

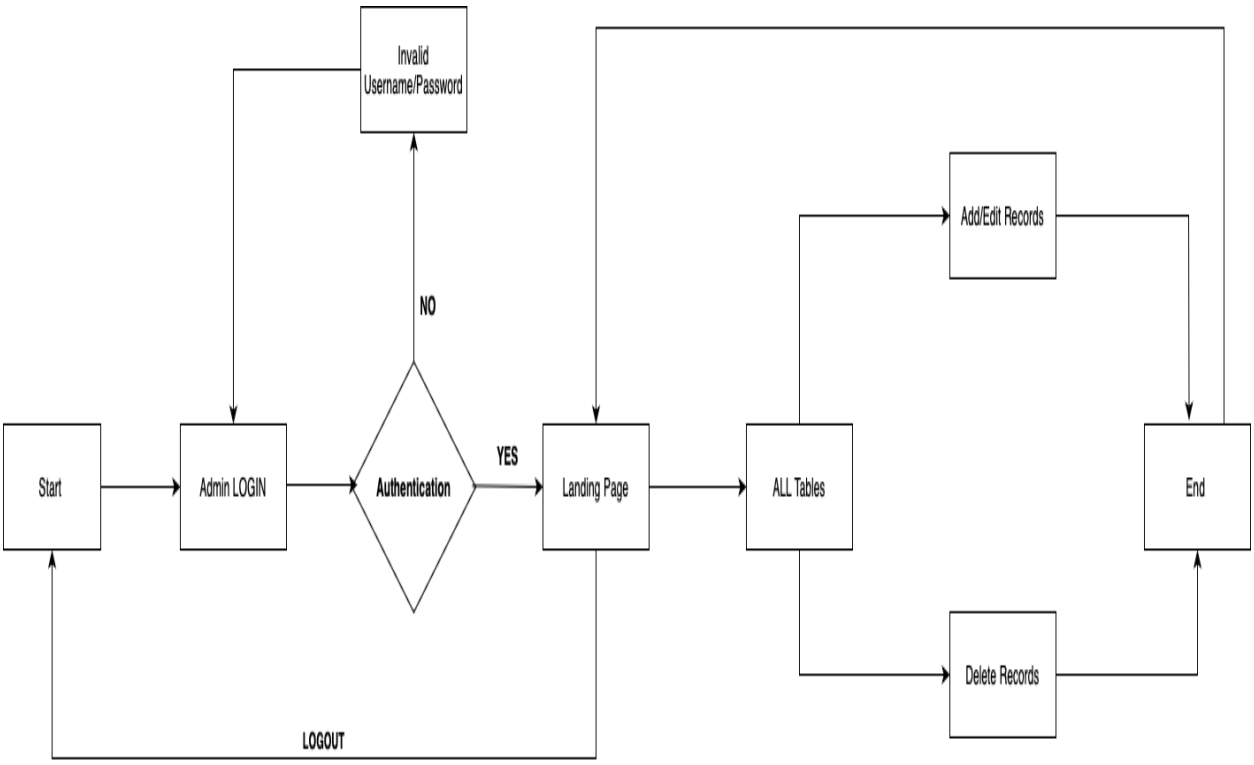



**Figure 11.1.1 GUE Admin POV**

## 11.1.2 GUE SHOPKEEPER POV

Warehouse Management System - User POV

1. **Start**
    - The system begins, initiating the user's interaction.
2. **User Login**
    - User (shopkeeper) enters credentials (username and password).
    - If invalid, the system shows an **"Invalid Username/Password"** error.

    - If the user is new, they are redirected to **Registration** for account creation.
3. **Authentication**
    - If the login is successful, the user is authenticated and directed to the **Landing Page**.
4. **Landing Page**
    - From here, the user can access various options, such as:
        - **Search Product**: The user can search for items in the warehouse.
        - **Product Catalog Screen**: After searching, the user sees a list of products.
        - **Shipment Tracking:** User can see their shipment and where is it
        - **Transaction History:** User can view all thor past transaction history
        - **Borrow Request Screen:** User can request the product from warehouse
5. **Product Catolog Screen:** The user can view the product's detail page before purchasing, which includes information about its category, location, stock availability, units, and price.

6. **Borrow Requests**
    - Users can make borrowing requests through a **Borrow Request Screen**, check the **Status of Requests**, and manage product reservations.
    - The user can add products to the cart and proceed to the **Payment** step.
    - If payment is unsuccessful, the user is returned to the cart to try again.
    - If payment is successful, the user moves to the **Order Management Screen**.
7. **Order Management**
    - The user can view order details on the **Order Management Screen**.

- This screen also allows tracking shipment statuses and viewing **Transaction History**.

8. **Logout or End**
    - The user can log out at any time, ending the session.

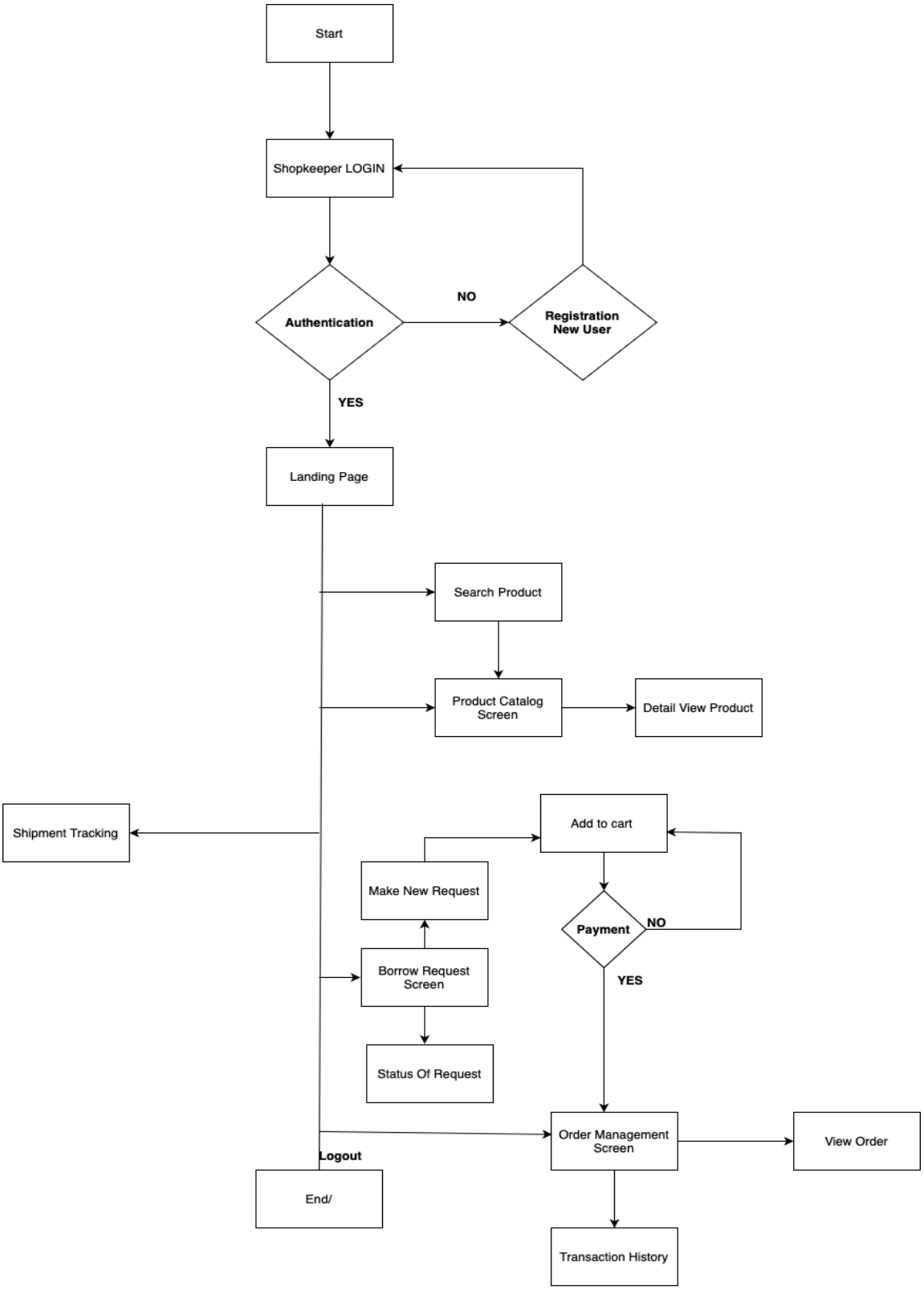



**Figure 11.1.2 GUE Shopkeeper POV**

# 11.2 VIEWS IMPLEMENTATION

### 1. vw_ProductInventory

This view provides a comprehensive overview of the product inventory, combining information from multiple tables:

- Includes product details such as name, category, size, and price
- Shows the current quantity in stock for each product
- Displays the supplier information for each product
- Includes warehouse and location details where the product is stored

This view is useful for inventory management, product analysis, and supply chain optimization.

### 2.vw_OrderSummary

The Order Summary view consolidates information about orders, shipments, and transactions:

- Presents key order details including date, quantity, status, and amount
- Includes the shopkeeper's name who placed the order
- Shows shipment status and tracking number if available
- Displays the payment method used for the transaction

This view is valuable for order tracking, sales analysis, and customer service inquiries.

### 3. vw_WarehouseCapacity

This view focuses on warehouse capacity and utilization:

- Provides warehouse identification and location details
- Shows the maximum capacity, current capacity, and available space for each warehouse
- Calculates the occupancy percentage for each warehouse

This view is essential for warehouse management, capacity planning, and logistics optimization.

**4. vw_ProductPerformance**

The Product Performance view analyzes sales data for each product:

- Includes product identification, name, and category
- Calculates total quantity sold and total revenue for each product
- Computes the average selling price per product

This view is crucial for sales analysis, product performance evaluation, and inventory planning.

**5. vw_SupplierPerformance**

This view assesses supplier performance based on the products they supply:

- Includes supplier identification and contact information
- Calculates the total number of unique products supplied by each supplier
- Shows the total inventory quantity supplied by each supplier

This view is useful for supplier relationship management, procurement strategy, and supply chain analysis.

**6. vw_BorrowRequestStatus**

The Borrow Request Status view provides information about product borrowing requests:

- Includes request details such as ID, date, and status
- Shows the product name associated with each request
- Displays the shopkeeper's name and email who made the request

This view is valuable for managing product loans, tracking request statuses, and coordinating with shopkeepers.

## 11.2.1 CODE OF VIEW IMPLEMENTATION

**1. NON UPDATABLE PART OF CODE**

```
-- View for Product Inventory
CREATE VIEW vw_ProductInventory AS
SELECT
    p.productID,
    p.productName,
    c.CategoryName,
    p.Size,
    p.PricePerUnit,
    p.QuantityInStock,
```

```
    s.SupplierName,
    w.WarehouseID,
    l.Country,
    l.State
FROM PRODUCT p
JOIN CATEGORY c ON p.CategoryId = c.CategoryID
JOIN SUPPLIER s ON p.SUPPLIER_SupplierID = s.SupplierID
JOIN STORAGE st ON p.productID = st.PRODUCT_productID
JOIN WAREHOUSE w ON st.WAREHOUSE_WarehouseID =
w.WarehouseID
JOIN LOCATION l ON w.LocationId = l.locationId;
```

```
-- View for Order Summary
CREATE VIEW vw_OrderSummary AS
SELECT
    o.OrderId,
    o.OrderDate,
    o.Quantity,
    o.OrderStatus,
    o.Amount,
    sk.FullName AS ShopkeeperName,
    s.ShipmentStatus,
    s.TrackingNumber,
    t.PaymentMethod
```

```
FROM `ORDER` o
JOIN SHOPKEEPER sk ON o.ShopkeeperId = sk.ShopkeeperID
LEFT JOIN SHIPMENT s ON o.OrderId = s.OrderId
LEFT JOIN TRANSACTION t ON o.OrderId = t.OrderId;

-- View for Warehouse Capacity
CREATE VIEW vw_WarehouseCapacity AS
SELECT
    w.WarehouseID,
    l.Country,
    l.State,
    l.ZipCode,
    w.MaxCapacity,
    w.CurrentCapacity,
    w.AvailableSpace,
    (w.CurrentCapacity * 100.0 / w.MaxCapacity) AS
OccupancyPercentage
FROM WAREHOUSE w
JOIN LOCATION l ON w.LocationId = l.locationId;

-- View for Product Performance
CREATE VIEW vw_ProductPerformance AS
SELECT
    p.productID,
```

```
    p.productName,
    c.CategoryName,
    SUM(o.Quantity) AS TotalQuantitySold,
    SUM(o.Amount) AS TotalRevenue,
    AVG(o.Amount / o.Quantity) AS AverageSellingPrice
FROM PRODUCT p
JOIN CATEGORY c ON p.CategoryId = c.CategoryID
JOIN `ORDER` o ON p.productID = o.OrderId
GROUP BY p.productID, p.productName, c.CategoryName;

-- View for Supplier Performance
CREATE VIEW vw_SupplierPerformance AS
SELECT
    s.SupplierID,
    s.SupplierName,
    s.ContactPerson,
    s.Email,
    COUNT(DISTINCT p.productID) AS
TotalProductsSupplied,
    SUM(p.QuantityInStock) AS TotalInventorySupplied
FROM SUPPLIER s
JOIN PRODUCT p ON s.SupplierID = p.SUPPLIER_SupplierID
GROUP BY s.SupplierID, s.SupplierName, s.ContactPerson,
s.Email;
```

## 2. UPDATABLE PART OF CODE

```
-- View for Borrow Request Status
CREATE VIEW vw_BorrowRequestStatus AS
SELECT
    br.RequestID,
    br.RequestDate,
    br.BorrowStatus,
    p.productName,
    sk.FullName AS ShopkeeperName,
    sk.Email AS ShopkeeperEmail
FROM BORROWREQUEST br
JOIN PRODUCT p ON br.ProductID = p.productID
JOIN SHOPKEEPER sk ON br.ShopkeeperID =
sk.ShopkeeperID;
```

# 12. REFERENCES

**[1] Zoho Inventory**

**[2] Fishbowl Inventory**

**[3] Korber Warehouse Management Systems**

**[4] Warehouse management system - ER Diagram**

**[5] Design ER Diagrams for Supply Chain Management**

**[6] Draw.io**

**[7] SQL data types**

**[8] String type syntax**

**[9] Char**

**[10] Binary and Varbinary**

**[11] Blob**

**[12] Enum**

**[13] Set**

**[14] JSON data types**

**[15] Column-indexes**

**[16] Optimization-indexes**

**[17] Insert-optimization**