

Warehouse Management System

Database Management System

MSCS 542L-256-24F

TechBoys



Marist College
School of Computer Science and Mathematics

Submitted To:
Dr. Reza Sadeghi

DEC 04, 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

1. DESCRIPTION OF TEAM MEMBERS	1
2. INTRODUCTION	2
3. PROJECT OBJECTIVE	3
4. REVIEW	4
5. MERITS	5
6. GITHUB REPOSITORY	6
7. ENTITY RELATIONSHIP MODEL (ER MODEL)	6
7.1 IMPLEMENTATION OF ER DIAGRAM	8
7.2 ENHANCED ENTITY RELATIONSHIP MODEL (EER MODEL)	13
9. DATABASE DEVELOPMENT	39
9.1 DESCRIPTION:	39
9.2 CREATE STATEMENTS	49
10.LOADING DATA AND PERFORMANCE ENHANCEMENT	58
10.1 HANDLING KEY CONSTRAINTS:	58
10.2 IMPORTING DATA:	62
10.3 OPTIMIZATION:	66
10.5 NORMALIZATION CHECK	77
11.APPLICATION DEVELOPMENT	81
11.1 USER EXPERIENCE DESIGN	81
11.1.1 UX ADMIN POV	81
11.1.2 UX SHOPKEEPER POV	83
11.2 VIEWS IMPLEMENTATION	87
11.2.1 CODE OF VIEW IMPLEMENTATION	89
12. SECURITY CONTROL MEASURES IN DATABASE SYSTEM	94
13.PASSWORD MANAGEMENT	114
14. STORAGE ENGINE	117

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

15. DATABASE INTEGRITY	124
16. GUI	132
17. REFERENCES	137

TABLE OF FIGURES

1. Figure 7.1 Relationship of Entity Relationship Model	07
2. Figure 7.2 External ER Shopkeeper POV	08
3. Figure 7.3 External ER Admin POV	09
4. Figure 7.4 Entity Relationship Diagram	11
5. Figure 7.5 Enhanced Entity Relationship	13
6. Figure 11.1.1 UX Admin POV.....	83
7. Figure 11.1.2 UX Shopkeeper POV.....	86

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

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

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:

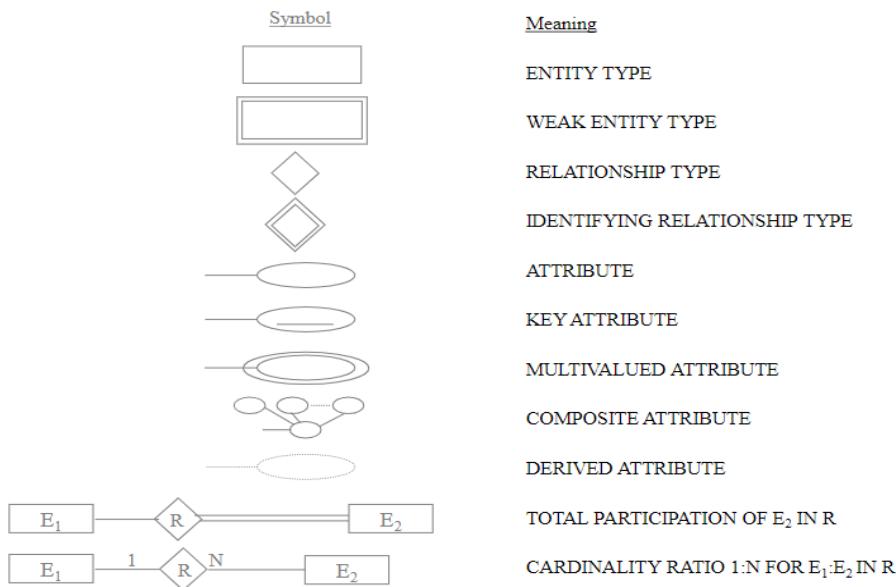


Figure 7. 1 Relationship of Entity Relationship Model

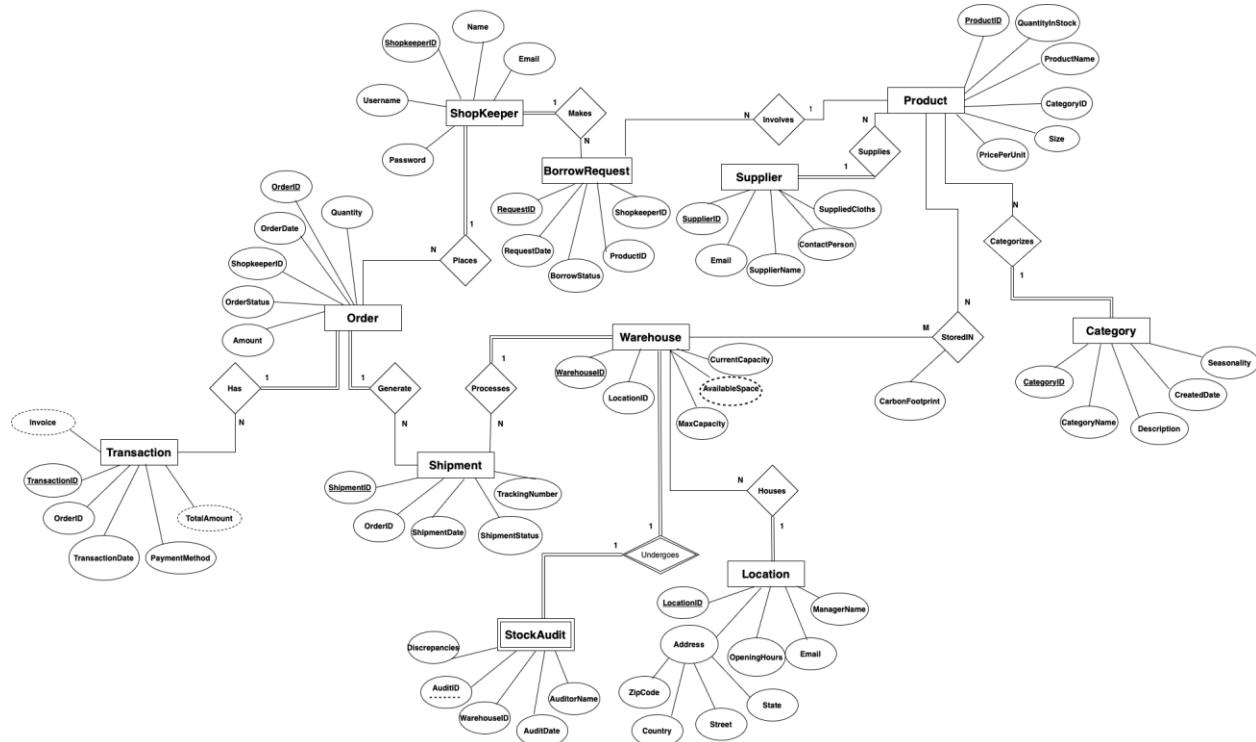


Figure 7.2 External ER SHOPKEEPER POV [6]

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.

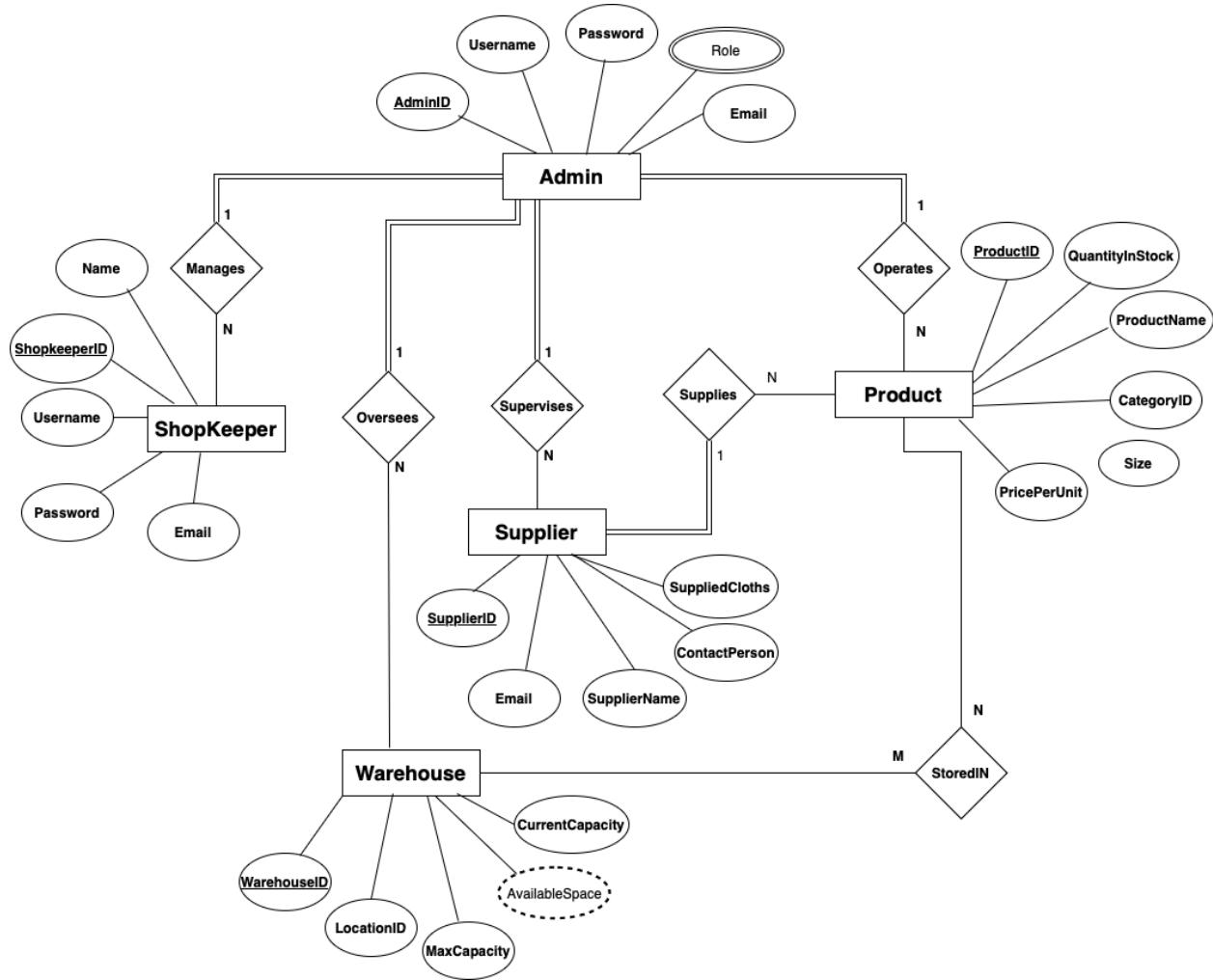


Figure 7. 3 External ER ADMIN POV [6]

The diagram places the **Admin** entity at the center, indicating a centralized management approach. The **Admin** has oversight of **ShopKeeper**, **Supplier**, and **Product**, 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.

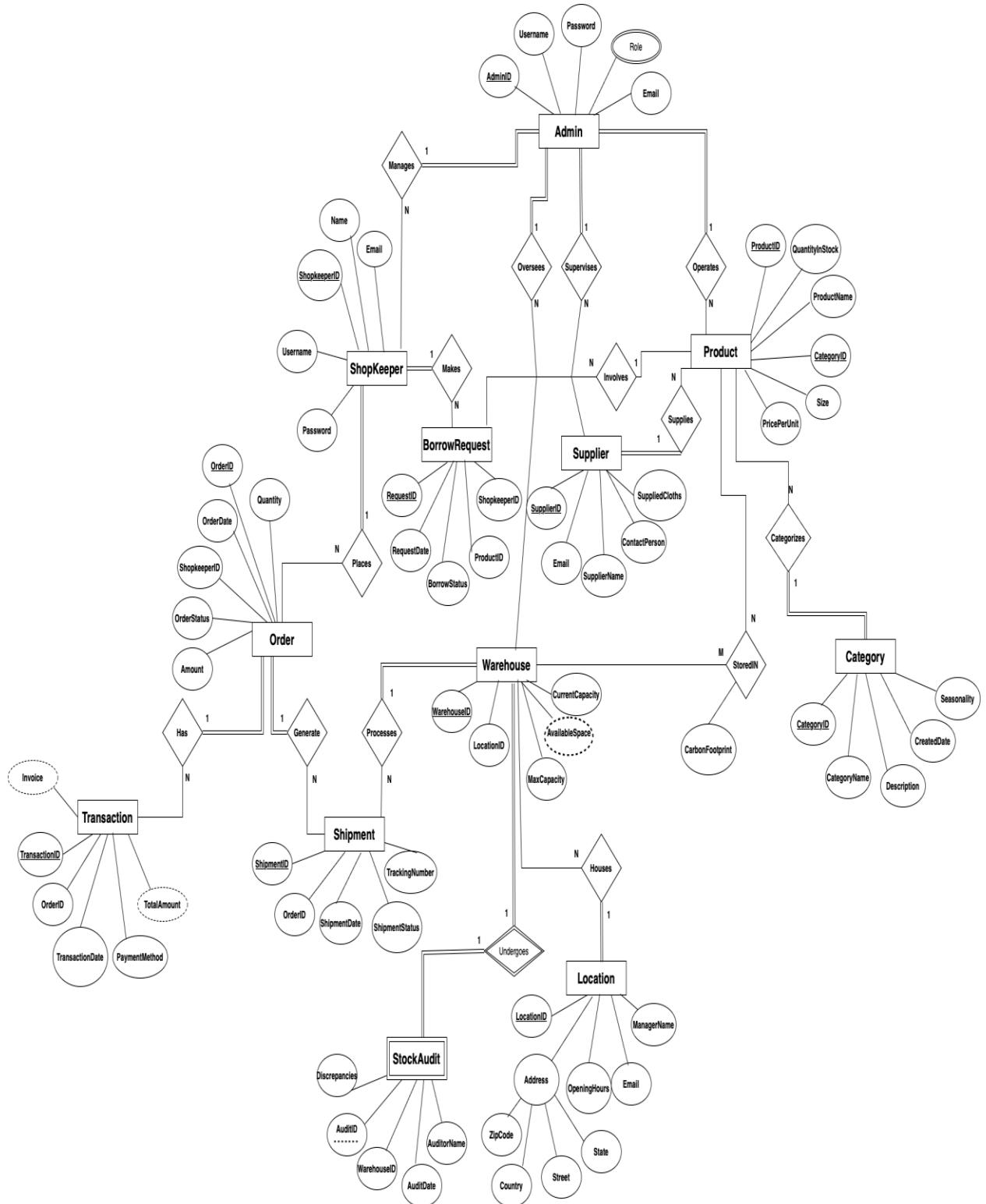


Figure 7.4 Entity Relationship Diagram [6]

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.

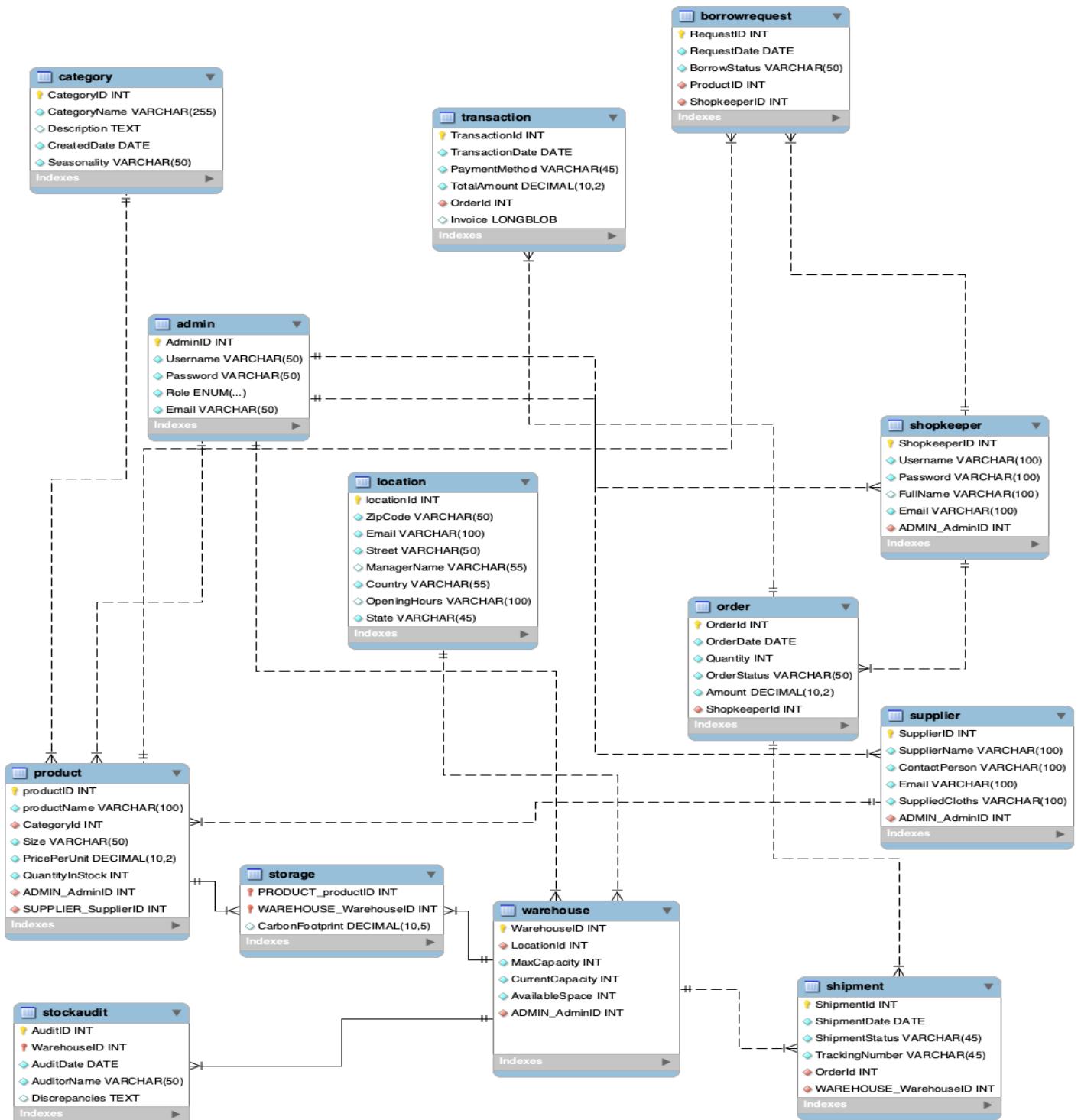


Figure 7.5 Enhanced Entity Relationship

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 bytes 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 commas.
- 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:



The screenshot shows the MySQL Workbench interface. On the left, there is a code editor with the following SQL script:

```

1 • Drop database if exists Phase3;
2
3 • CREATE DATABASE Phase3;
4
5 • USE Phase3;
6
7 • CREATE TABLE admin (
8     username VARCHAR(50), -- variable length string
9     age INT,
10    location VARCHAR(100),
11    email VARCHAR(100),
12    profile_picture BLOB, -- Binary Large Objects (e.g., images)
13    document VARBINARY(65535), -- variable length binary data (e.g., files)
14    bio TEXT, -- TEXT can accommodate large text data
15    status ENUM('Active', 'Inactive', 'Pending'), -- predefined set of values
16    preferences SET('Email', 'SMS', 'Push Notification'), -- SET is used for a collection of values
17    unique_id BINARY(16) -- fixed length binary data
18 );
19
20 • INSERT INTO admin (username, age, location, email, profile_picture, document, bio, status, preferences, unique_id)

```

A tooltip on the right side of the interface reads: "Automatic context help is disabled. Use the toolbar manually get help for the current caret position or toggle automatic help."

At the bottom, the "Output" tab displays the execution results:

#	Action	Time	Message	Duration / Fetch
1	Drop database if exists Phase3	13:12:29	1 row(s) affected	0.015 sec
2	CREATE DATABASE Phase3	13:12:29	1 row(s) affected	0.000 sec
3	USE Phase3	13:12:29	0 row(s) affected	0.000 sec
4	INSERT INTO admin (username, age, location, email, profile_picture, document, bio, status, preferences, unique_id)	13:12:29	0 row(s) affected	0.016 sec

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

```
20 • INSERT INTO admin (username, age, location, email, profile_picture, document, bio, status, preferences, unique_values (
21     'Sumanth',
22     24,
23     'New York',
24     'sumanth@gmail.com',
25     '0x89504E47000A10A00000000494844520000010000001008060000001F4FB061', -- Example BLOB data
26     '0x255044620312E35000A25E2E3CFD300A312030206F626A00A3C3C2F4C656E6774682034', -- Example VARBINARY data
27     'Intro of sumanth', -- Example TEXT data
28     'Active', -- Status ENUM
29     'Email,SMS', -- Preferences SET
30     UNHEX('4F6A1F8E9D6F4E888C9A8F2C5D4E6F7D') -- Example BINARY data
31 );
32
33
34 • INSERT INTO admin (username, age, location, email, profile_picture, document, bio, status, preferences, unique_values (
35     'Abhijeet',
36     24,
37     'New York'
38 ));
```

Output

Action Output

#	Time	Action	Message	Duration / Fetc
1	13:13:36	INSERT INTO admin (username, age, location, email, profile_picture, document, bio, status, preferences, unique_values)	1 row(s) affected	0.016 sec
2	13:13:36	INSERT INTO admin (username, age, location, email, profile_picture, document, bio, status, preferences, unique_values)	1 row(s) affected	0.000 sec
3	13:13:36	INSERT INTO admin (username, age, location, email, profile_picture, document, bio, status, preferences, unique_values)	1 row(s) affected	0.000 sec

Context Help Snippets

Automatic context help is disabled. Use the toolbar manually get help for the current caret position or toggle automatic help.

String Functions:

1. **LENGTH ()**: Provides length of string

```
64 • SELECT username, LENGTH(username) AS username_length FROM admin; -- Length of string
A5
Result Grid | Filter Rows: Export: Wrap Cell Content: □
username username_length
Sumanth 7
Abhjeet 8
Sagar 5
Result Grid
Form Editor
Field Types
Result 2 x Read Only Context Help Snippets
Output
Action Output
# Time Action Message Duration / Fetch
1 13:17:16 SELECT username, LENGTH(username) AS username_length FROM admin LIMIT 0, 1000 3 row(s) returned 0.000 sec / 0.000
```

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

```

66 •   SELECT username, LOWER(location) AS lower_location, UPPER(email) AS upper_email FROM admin; -- Converts the s
67
68 •   SELECT CONCAT(username, ' : ', age) AS user_info FROM admin; -- Concatenates multiple strings together
69
70 •   SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin; -- retrieves substring from string
71

```

Result Grid

username	lower_location	upper_email
Sumanth	new york	SUMANTH@GMAIL.COM
Abhijeet	new york	ABHI@GMAIL.COM
Sagar	new york	SAGAR@GMAIL.COM

Action Output

#	Time	Action	Message
1	13:18:11	SELECT username, LOWER(location) AS lower_location, UPPER(email) AS upper_email FROM admin LIMIT 0, ...	3 row(s) returned

3. Concat: Concatenates multiple strings together

```

68 •   SELECT CONCAT(username, ' : ', age) AS user_info FROM admin; -- Concatenates multiple strings together
69
70 •   SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin; -- retrieves substring from string
71

```

Result Grid

user_info
Sumanth : 24
Abhijeet : 24
Sagar : 27

Action Output

#	Time	Action	Message
1	13:18:11	SELECT username, LOWER(location) AS lower_location, UPPER(email) AS upper_email FROM admin LIMIT 0, ...	3 row(s) returned
2	13:18:42	SELECT CONCAT(username, ' : ', age) AS user_info FROM admin LIMIT 0, 1000	3 row(s) returned

4. Substring (): Retrieves substring from string

```

70 •   SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin; -- retrieves substring from string
71
72 •   SELECT username, INSTR(email, 'gmail') AS gmail_position FROM admin; -- finds the position of the first occur
73
74 •   SELECT username, REPLACE(location, 'New York', 'NY') AS short_location FROM admin; -- replacing string

```

Result Grid

username	email_part
Sumanth	suman
Abhijeet	abhi@
Sagar	sagar

Action Output

#	Time	Action	Message
1	13:18:11	SELECT username, LOWER(location) AS lower_location, UPPER(email) AS upper_email FROM admin LIMIT 0, ...	3 row(s) returned
2	13:18:42	SELECT CONCAT(username, ':', age) AS user_info FROM admin LIMIT 0, 1000	3 row(s) returned
3	13:19:05	SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin LIMIT 0, 1000	3 row(s) returned

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

```

74 •   SELECT username, REPLACE(location, 'New York', 'NY') AS short_location FROM admin; -- replacing string
75
76 •   SELECT username, TRIM(both ' ' FROM bio) AS trimmed_bio FROM admin; -- removes spaces from the string
77
78
79 -- Search

```

Result Grid

username	short_location
Sumanth	NY
Abhijeet	NY
Sagar	NY

Action Output

#	Time	Action	Message
1	13:18:11	SELECT username, LOWER(location) AS lower_location, UPPER(email) AS upper_email FROM admin LIMIT 0, ...	3 row(s) returned
2	13:18:42	SELECT CONCAT(username, ':', age) AS user_info FROM admin LIMIT 0, 1000	3 row(s) returned
3	13:19:05	SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin LIMIT 0, 1000	3 row(s) returned
4	13:19:34	SELECT username, REPLACE(location, 'New York', 'NY') AS short_location FROM admin LIMIT 0, 1000	3 row(s) returned

6. TRIM (): Removes spaces from the string

```

76 •  SELECT username, TRIM(both ' ' FROM bio) AS trimmed_bio FROM admin; -- removes spaces from the string
77
78
79 -- Search

```

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

username	trimmed_bio
Sumanth	Intro of sumanth
Abhijeet	Intro of Abhijeet
Sagar	Intro of Sagar

Form Editor

Field Types

Result 7 x Read Only Context Help Snip

Action Output

#	Time	Action	Message
1	13:18:11	SELECT username, LOWER(location) AS lower_location, UPPER(email) AS upper_email FROM admin LIMIT 0, ...	3 row(s) returned
2	13:18:42	SELECT CONCAT(username, ':', age) AS user_info FROM admin LIMIT 0, 1000	3 row(s) returned
3	13:19:05	SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin LIMIT 0, 1000	3 row(s) returned
4	13:19:34	SELECT username, REPLACE(location, 'New York', 'NY') AS short_location FROM admin LIMIT 0, 1000	3 row(s) returned
5	13:19:49	SELECT username, TRIM(both '' FROM bio) AS trimmed_bio FROM admin LIMIT 0, 1000	3 row(s) returned

7. Searching:

```

81 •  SELECT * FROM admin WHERE email LIKE '%gmail%';

```

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

username	age	location	email	profile_picture	document	bio	status	preferences	unique_id
Sumanth	24	New York	sumanth@gmail.com	BLOB	BLOB	Intro of sumanth	Active	Email,SMS	BLOB
Abhijeet	24	New York	abhi@gmail.com	BLOB	BLOB	Intro of Abhijeet	Pending	Email	BLOB
Sagar	27	New York	sagar@gmail.com	BLOB	BLOB	Intro of Sagar	Active	SMS	BLOB

Form Editor

Field Types

admin 8 x Read Only Context

Action Output

#	Time	Action	Message
1	13:18:11	SELECT username, LOWER(location) AS lower_location, UPPER(email) AS upper_email FROM admin LIMIT 0, ...	3 row(s) returned
2	13:18:42	SELECT CONCAT(username, ':', age) AS user_info FROM admin LIMIT 0, 1000	3 row(s) returned
3	13:19:05	SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin LIMIT 0, 1000	3 row(s) returned
4	13:19:34	SELECT username, REPLACE(location, 'New York', 'NY') AS short_location FROM admin LIMIT 0, 1000	3 row(s) returned
5	13:19:49	SELECT username, TRIM(both '' FROM bio) AS trimmed_bio FROM admin LIMIT 0, 1000	3 row(s) returned
6	13:20:20	SELECT * FROM admin WHERE email LIKE '%gmail%' LIMIT 0, 1000	3 row(s) returned

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

```
83 •   SELECT * FROM admin WHERE FIND_IN_SET('Email', preferences) > 0;
```

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

username	age	location	email	profile_picture	document	bio	status	preferences	unique_id
Sumanth	24	New York	sumanthal@gmail.com	<input type="button" value="BLOB"/>	<input type="button" value="BLOB"/>	Intro of sumanthal	Active	Email,SMS	<input type="button" value="BLOB"/>
Abhijeet	24	New York	abhi@gmail.com	<input type="button" value="BLOB"/>	<input type="button" value="BLOB"/>	Intro of Abhijeet	Pending	Email	<input type="button" value="BLOB"/>

Result Grid | Form Editor | Field Types

admin 9 × Read Only

Output:

Action Output

#	Time	Action	Message
2	13:18:42	SELECT CONCAT(username, ':', age) AS user_info FROM admin LIMIT 0, 1000	3 row(s) returned
3	13:19:05	SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin LIMIT 0, 1000	3 row(s) returned
4	13:19:34	SELECT username, REPLACE(location, 'New York', 'NY') AS short_location FROM admin LIMIT 0, 1000	3 row(s) returned
5	13:19:49	SELECT username, TRIM(both '' FROM bio) AS trimmed_bio FROM admin LIMIT 0, 1000	3 row(s) returned
6	13:20:20	SELECT * FROM admin WHERE email LIKE '%gmail%' LIMIT 0, 1000	3 row(s) returned
7	13:21:01	SELECT * FROM admin WHERE FIND_IN_SET('Email', preferences) > 0 LIMIT 0, 1000	2 row(s) returned

8. Extracting values using SELECT:

```
85 -- Extracting values using SELECT
86
87 •   SELECT username, LOWER(email) AS lowercase_email, SUBSTRING(location, 1, 3) AS short_location
88   FROM admin
89   WHERE status = 'Active'
90   AND email LIKE '%gmail%';
```

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

username	age	location	email	profile_picture	document	bio	status	preferences	unique_id
Sumanth	24	New York	sumanthal@gmail.com	<input type="button" value="BLOB"/>	<input type="button" value="BLOB"/>	Intro of sumanthal	Active	Email,SMS	<input type="button" value="BLOB"/>
Abhijeet	24	New York	abhi@gmail.com	<input type="button" value="BLOB"/>	<input type="button" value="BLOB"/>	Intro of Abhijeet	Pending	Email	<input type="button" value="BLOB"/>

Result Grid | Form Editor | Field Types

admin 9 × Read Only

Output:

Action Output

#	Time	Action	Message
2	13:18:42	SELECT CONCAT(username, ':', age) AS user_info FROM admin LIMIT 0, 1000	3 row(s) returned
3	13:19:05	SELECT username, SUBSTRING(email, 1, 5) AS email_part FROM admin LIMIT 0, 1000	3 row(s) returned
4	13:19:34	SELECT username, REPLACE(location, 'New York', 'NY') AS short_location FROM admin LIMIT 0, 1000	3 row(s) returned
5	13:19:49	SELECT username, TRIM(both '' FROM bio) AS trimmed_bio FROM admin LIMIT 0, 1000	3 row(s) returned
6	13:20:20	SELECT * FROM admin WHERE email LIKE '%gmail%' LIMIT 0, 1000	3 row(s) returned
7	13:21:01	SELECT * FROM admin WHERE FIND_IN_SET('Email', preferences) > 0 LIMIT 0, 1000	2 row(s) returned

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()');
26

```

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

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

```
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) FROM json_demo_table
42
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 JSON object') 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", "value4"), 'Insertion using JSON_OBJECT()') 1 row(s) affected
5 14:41:44 INSERT INTO json_demo_table(json_data, description) VALUES (JSON_ARRAY("string2", 3, '{"key5":"value5","key6":"value6"}'), 'Insertion using JSON_ARRAY()') 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
39
40
41
Output
Action Output
# Time Action Message
2 14:40:57 INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1":"value1","key2":"value2"}', 'Direct JSON object') 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", "value4"), 'Insertion using JSON_OBJECT()') 1 row(s) affected
5 14:41:44 INSERT INTO json_demo_table(json_data, description) VALUES (JSON_ARRAY("string2", 3, '{"key5":"value5","key6":"value6"}'), 'Insertion using JSON_ARRAY()') 1 row(s) affected
6 14:42:12 INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1":["string3", 6], "key2":"value2"}', 'Nested JSON: array in object') 1 row(s) affected
```

```

35 -- Display the contents of the table
36 • SELECT * FROM json_demo_table;
37
38

```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
json_data	description			
["key1": "value1", "key2": "value2"]	Direct JSON object insertion			
[3, "string1"]	Direct JSON array insertion			
{"key3": "value3", "key4": "value4"} ["string2", 3, {"\key5": "value5", "\key6": "v... {"key1": ["string3", 6], "key2": "value2"}]	Insertion using JSON_OBJECT() Insertion using JSON_ARRAY() Nested JSON: array in object			

json_demo_table10 ×

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": "value5", "key6": "value6"}))	1 row(s) affected
6	14:42:12	INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1": ["string3", 6], "key2": "value2"}', 'Nest...')	1 row(s) affected
7	14:42:23	SELECT * FROM json_demo_table LIMIT 0, 1000	5 row(s) returned

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

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

```

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, '{"keymerge": "valuemerge"}')
42     JSON_MERGE_PATCH(json_data, '[1, 2]'),JSON_MERGE_PATCH('[1, 2]', json_data),JSON_MERGE_PATCH(json_data, '{"keymerge": "valuemerge"}')
43   FROM json_demo_table;
44

```

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

ta	JSON_MERGE_PRESERVE('{"keymerge": "valuemerge"}', json_data)	JSON_MERGE_PATCH(json_data, '[1, 2]')	JSON_MERGE_PATCH('[1, 2]', json_data)	JSON_MERGE_PATCH(json_data, '{"keymerge": "valuemerge"}')
▶ , "keymerge": ...	{"key1": "value1", "key2": "value2", "keymerge": ... [1, 2]}	{"key1": "value1", "key2": "value2"}	{[3, "string1"]}	{"key1": "value1", "key2": "value2"}
, "keymerge": ...	[{"keymerge": "valuemerge"}, 3, "string1"]	[1, 2]	{"key3": "value3", "key4": "value4"}	{"key3": "value3", "key4": "value4"}
, "keymerge": ...	{"key3": "value3", "key4": "value4", "keymerge": ... [1, 2]}	[{"key5": "value5", "key6": "value6"}]	[{"key2": 3, {"key5": "value5"}, {"key6": "value6"}]	{"key3": "value3", "key4": "value4"}
\key6": "value6"];	["keymerge": "valuemerge"], "string2", 3, {"\key... [1, 2]			{"key1": ["string3", 6], "key2": "value2"}
ue2", "keym...	{"key1": ["string3", 6], "key2": "value2", "keym... [1, 2]			{"key1": ["string3", 6], "key2": "value2"}
ue2", "keym...	{"key1": ["string3", 6], "key2": "value2", "keym... [1, 2]			{"key1": ["string3", 6], "key2": "value2"}

Result 11 ×

Output

Action Output

#	Time	Action	Message
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", "value4", "key5", "value5", "key6", "value6"))	1 row(s) affected
5	14:41:44	INSERT INTO json_demo_table(json_data, description) VALUES (JSON_ARRAY("string2", 3, {"key5": "value5", "key6": "value6"}))	1 row(s) affected
6	14:42:12	INSERT INTO json_demo_table(json_data, description) VALUES ([{"key1": ["string3", 6], "key2": "value2"}])	1 row(s) affected
7	14:42:23	SELECT * FROM json_demo_table LIMIT 0, 1000	5 row(s) returned
8	14:43:38	SELECT JSON_MERGE_PRESERVE(json_data, '[1, 2]),JSON_MERGE_PRESERVE('[1, 2]', json_data),JSO...	5 row(s) returned

JSON Functions:

1. JSON_ARRAY_APPEND ():

```

47    -- Appending integer element into an array at index 1
48 •   SELECT json_data, JSON_ARRAY_APPEND(json_data, '$[1]', 1)
49   FROM json_demo_table;
50
51    -- Inserting integer element into an array at index 0

```

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

json_data	JSON_ARRAY_APPEND(json_data, '\$[1]', 1)
▶ {"key1": "value1", "key2": "value2"} [3, "string1"]	{"key1": "value1", "key2": "value2"} [3, ["string1", 1]]
{"key3": "value3", "key4": "value4"} ["string2", 3, {"\key5": "value5"}, {"key6": "value6"}]	{"key3": "value3", "key4": "value4"} ["string2", [3, 1], {"\key5": "value5"}, {"key6": "value6"}]
{"key1": ["string3", 6], "key2": "value2"}	{"key1": ["string3", 6], "key2": "value2"}

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": "\v...} {"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": "value2", "key6": "value3"}, [{"key1": "value1", "key2": "value2"}, {"key3": "value3", "key4": "value4"}]), 'New JSON object')
6	14:42:12	INSERT INTO json_demo_table(json_data, description) VALUES ({"key1": ["string3", 6], "key2": "value2"}, 'New JSON object')
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),JSON_MERGE(json_data, [1, 2])
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"} [3, "string1"] {"key3": "value3", "key4": "value4"} ["string2", 3, {"\key5": "\value5", "\key6": "\v...} {"key1": ["string3", 6], "key2": "value2"}	41 19 41 56 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 pairs

```

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");

```

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"} [3, "string1"] {"key3": "value3", "key4": "value4"} ["string2", 3, {"\\"key5\\": "value5", "\\"key6\\": "v... {"key1": ["string3", 6], "key2": "value2"}	{"key1": "value1", "key2": "value2", "keynew": ... [3, "string1"] {"key3": "value3", "key4": "value4", "keynew": ... ["string2", 3, {"\\"key5\\": "value5", "\\"key6\\": "v... {"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'} [3, "string1"] {'key3': 'value3', 'key4': 'value4'} ['string2', 3, {"\key5": "value5", "\key6": "\v... {'key1': ['string3', 6], 'key2': 'value2'}	{"key1": "value1", "key2": "value2"} ["string1"] {"key3": "value3", "key4": "value4"} [3, {"\key5": "value5", "\key6": "value6"}] {"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'} [3, "string1"] {'key3': 'value3', 'key4': 'value4'} ['string2', 3, {"\key5": "value5", "\key6": "\v... {'key1': ['string3', 6], 'key2': 'value2'}	{"key1": "val3", "key2": "value2"} [3, "string1"] {"key3": "value3", "key4": "value4"} [3, {"\key5": "value5", "\key6": "value6"}] {"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	
	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	<p>Usage: Represents administrators responsible for managing warehouses, products, shopkeepers, and suppliers.</p> <p>Data attributes:</p> <ol style="list-style-type: none"> 1. AdminID (Integer, Primary Key, Not Null) 2. Username (Varchar (50),Not Null) 3. Password (Varchar (50),Not Null) 4. Email (Varchar (50),Not Null) 5. Role (Enum (Manager, Supervisor),Not Null) <p>Constraints:</p> <ol style="list-style-type: none"> 1. AdminID must be unique and cannot be null. PRIMARY KEY 2. Email must follow a valid email format. 3. Role is an Enum (Manager, Supervisor) <p>Relationships :</p>

	<ol style="list-style-type: none"> 1. Admin manages multiple Warehouses 2. Admin manages multiple Products, 3. Admin manages multiple Shopkeepers 4. Admin manages multiple Suppliers.
Product	<p>Usage: Represents items that are stored in warehouses, supplied by suppliers, and categorized into various product categories.</p> <p>Data Attributes:</p> <ol style="list-style-type: none"> 1. ProductID (Integer, Primary Key, Not Null) 2. ProductName (Varchar (50), Not Null) 3. Size (Varchar(50), Not Null) 4. PricePerunit (Decimal(10,2),, Not Null) 5. QuantityInStock (Integer, Not Null) 6. CategoryID (Integer, Foreign Key, Not Null) 7. SupplierID (Integer, Foreign Key, Not Null) 8. AdminID(Integer, Foreign Key, Not Null) <p>Constraints:</p> <ol style="list-style-type: none"> 1. ProductID must be unique and cannot be null. PRIMARY KEY 2. CategoryID references the Category table. Foreign Key, 3. SupplierID references the Supplier table. Foreign Key,

	<p>4. AdminID references the admin table. Foreign Key</p> <p>Relationships:</p> <ol style="list-style-type: none"> 1. Product can be involved in multiple BorrowRequests . 2. Product Belongs to one Category. 3. Product Supplied by one Supplier. 4. Each product has an admin
--	---

Warehouse	<p>Usage: Represents storage locations for products and handles stock audits and shipments.</p> <p>Data Attributes:</p> <ol style="list-style-type: none"> 1. WarehouseID (Integer, Primary Key, Not Null) 2. LocationID (Integer, Foreign Key, Not Null) 3. MaxCapacity (Integer, Not Null) 4. AvailableSpace (Integer, Not Null) 5. CurrentCapacity (Integer, Not Null) 6. AdminID (references the admin table. Foreign Key, Not Null) <p>Constraints :</p> <ol style="list-style-type: none"> 1. WarehouseID must be unique and cannot be null. 2. LocationID references the Location table. <p>Relationships:</p> <ol style="list-style-type: none"> 1. Warehouse Stores multiple Storage. 2. Warehouse Processes multiple Shipments. 3. Warehouse Undergoes one StockAudit .
------------------	--

Supplier	<p>Usage: Represents entities that supply products to the warehouses.</p> <p>Data Attributes:</p> <ol style="list-style-type: none">1. SupplierID (Integer, Primary Key, Not Null)2. SupplierName (Varchar (100), Not Null)3. ContactPerson (Varchar (100), Not Null)4. Email (Varchar (100), Not Null)5. SuppliedCloths (Varchar (100), Not Null)6. AdminID (references the admin table. Foreign Key, Not Null) <p>Constraints:</p> <ol style="list-style-type: none">1. SupplierID must be unique and cannot be null.2. AdminID references the admin table. Foreign Key <p>Relationships:</p> <ol style="list-style-type: none">1. Supplier supplies multiple Products .2. Each Supplier has a admin
-----------------	---

StockAudit	<p>Usage: Represents audits performed on the stock of a warehouse.</p> <p>Data Attributes:</p> <ul style="list-style-type: none">1. AuditID (Integer, Primary Key, Not Null)2. WarehouseID (Integer, Foreign Key, Not Null)3. AuditDate (Date, Not Null)4. AuditorName (Varchar(50),Not Null)5. Discrepancies TEXT <p>Constraints:</p> <ul style="list-style-type: none">1. StockAuditID must be unique and cannot be null.2. WarehouseID references the Warehouse table. <p>Relationships:</p> <ul style="list-style-type: none">1. StockAuditID Performed on one Warehouse.
-------------------	--

Location	<p>Usage: Represents physical locations of warehouses or shops.</p> <p>Data Attributes:</p> <ol style="list-style-type: none"> 2. LocationID (Integer, Primary Key, Not Null) 3. Zipcode(Varchar(55),Not Null) 4. Email (Varchar (100),Not Null) 5. Street (Varchar (55),Not Null) 6. State (Varchar (55),Not Null) 7. Country (Varchar (55)Not Null) 8. OpeningHours(Varchar(55),) 9. ManagerName(Varchar(55),) <p>Constraints:</p> <ol style="list-style-type: none"> 1. LocationID must be unique and cannot be null. <p>Relationships:</p> <ol style="list-style-type: none"> 1. Location Contains multiple Warehouses .
Storage	<p>Usage : Represents the many-to-many relationship between Products and Warehouses, indicating which products are stored in which warehouses.</p> <p>Data Attributes:</p> <ol style="list-style-type: none"> 1. ProductID (Integer, Foreign Key, Not Null) 2. WarehouseID (Integer, Foreign Key, Not Null) 3. CarbonfootPrint (Double(10,5)) <p>Constraints:</p> <ol style="list-style-type: none"> 1. ProductID references the Product table and cannot be null. 2. WarehouseID references the Warehouse table and cannot be null. 3. Together, ProductID and WarehouseID form a composite primary key.

	<p>Relationships:</p> <ol style="list-style-type: none"> 1. Storage References for one Product. 2. Storage References one Warehouse.
Category	<p>Usage: Represents the category of products that are stored in warehouses and supplied by suppliers.</p> <p>Data Attributes:</p> <ol style="list-style-type: none"> 1. CategoryID (Integer, Primary Key, Not Null) 2. CategoryName (Varchar(255), Not Null) 3. Description (Text, Nullable) 4. CreatedDate (Date, Not Null) 5. Seasonality (Varchar(50), Nullable) <p>Constraints:</p> <ol style="list-style-type: none"> 1. CategoryID must be unique and cannot be null. PRIMARY KEY. <p>Relationships:</p> <ol style="list-style-type: none"> 1. CategoryID is referenced by the Product table. 2. Each Category can have multiple products.

BorrowRequest	<p>Usage: Represents the request made by shopkeepers to borrow products from warehouses.</p> <p>Data Attributes:</p> <ol style="list-style-type: none"> 1. RequestID (Integer, Primary Key, Not Null) 2. RequestDate (Date, Not Null) 3. BorrowStatus (Varchar(50), Not Null) 4. ProductID (Integer, Foreign Key, Not Null) 5. ShopkeeperID (Integer, Foreign Key, Not Null) <p>Constraints:</p> <ol style="list-style-type: none"> 1. RequestID must be unique and cannot be null. PRIMARY KEY. 2. ProductID references the Product table. 3. ShopkeeperID references the Shopkeeper table. <p>Relationships:</p> <ol style="list-style-type: none"> 1. Each BorrowRequest is associated with one Product. 2. Each BorrowRequest is associated with one Shopkeeper.
Transaction	<p>Usage: Represents financial transactions related to orders placed by shopkeepers.</p> <p>Data Attributes:</p> <ol style="list-style-type: none"> 1. TransactionId (Integer, Primary Key, Not Null) 2. TransactionDate (Date, Nullable) 3. PaymentMethod (Varchar(45), Nullable) 4. TotalAmount (Decimal(10,2), Nullable) 5. OrderId (Integer, Foreign Key, Nullable) 6. Invoice (Long BLOB, Nullable) <p>Constraints:</p> <ol style="list-style-type: none"> 1. TransactionId must be unique and cannot be null. PRIMARY KEY. 2. OrderId references the Order table. <p>Relationships:</p>

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

Order	<p>Usage: Represents orders placed by shopkeepers for products from warehouses.</p> <p>Data Attributes:</p> <ol style="list-style-type: none">1. OrderId (Integer, Primary Key, Not Null)2. OrderDate (Date, Not Null)3. Quantity (Integer, Not Null)4. OrderStatus (Varchar(50), Not Null)5. Amount (Decimal(10,2), Not Null)6. ShopkeeperId (Integer, Foreign Key, Not Null) <p>Constraints:</p> <ol style="list-style-type: none">1. OrderId must be unique and cannot be null. PRIMARY KEY.2. ShopkeeperId references the Shopkeeper table. <p>Relationships:</p> <ol style="list-style-type: none">1. Each Order is associated with one Shopkeeper.2. Each Order can have multiple Shipments.
--------------	---

Shipment	<p>Usage: Represents shipments for the orders placed by shopkeepers, tracking the shipment process.</p> <p>Data Attributes:</p> <ol style="list-style-type: none"> 1. ShipmentId (Integer, Primary Key, Not Null) 2. ShipmentDate (Date, Not Null) 3. ShipmentStatus (Varchar(45), Not Null) 4. TrackingNumber (Varchar(45), Not Null) 5. OrderId (Integer, Foreign Key, Not Null) 6. WAREHOUSE_WarehouseID (Integer, Foreign Key, Not Null) <p>Constraints:</p> <ol style="list-style-type: none"> 1. ShipmentId must be unique and cannot be null. PRIMARY KEY. 2. OrderId references the Order table. 3. WAREHOUSE_WarehouseID references the Warehouse table. <p>Relationships:</p> <ol style="list-style-type: none"> 1. Each Shipment is linked to one Order. 2. Each Shipment is processed by one Warehouse.
Shopkeeper	<p>Usage: Represents shopkeepers who place orders and borrow products from warehouses.</p> <p>Data Attributes:</p> <ol style="list-style-type: none"> 1. ShopkeeperID (Integer, Primary Key, Not Null) 2. Username (Varchar(100), Not Null) 3. Password (Varchar(100), Not Null) 4. FullName (Varchar(100)) 5. Email (Varchar(100), Not Null) 6. ADMIN_AdminID (Integer, Foreign Key, Not Null) <p>Constraints:</p> <ol style="list-style-type: none"> 1. ShopkeeperID must be unique and cannot be null. PRIMARY KEY. 2. ADMIN_AdminID references the Admin table. <p>Relationships:</p> <ol style="list-style-type: none"> 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` (
```

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

```
`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

	AdminID	Username	Password	Role	Email	
	1	Sumanth	password123	Manager	sumanth@gmail.com	
	NULL	NULL	NULL	NULL	NULL	

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
epperID)
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

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

```
FIELDS TERMINATED BY ','  
LINES TERMINATED BY '\n';
```

6	michael_d davismichi	Supervisor	michael.davis@yahoo.com
7	emily_wils wilsonemil	Manager	emily.wilson@gmail.com
8	david_milli millerdavic	Supervisor	david.miller@yahoo.com
9	jessica_gai garciajessi	Manager	jessica.garcia@gmail.com
10	kevin_rodr rodriguezk	Supervisor	kevin.rodriguez@yahoo.com
11	ashley_ma martinezat	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 jacksonanaj	Manager	angela.jackson@gmail.com
16	daniel_har harrisdanik	Supervisor	daniel.harris@yahoo.com
17	amanda_lk lewisamar	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 adamsreb	Manager	rebecca.adams@gmail.com
22	patrick_ne nelsonpatr	Supervisor	patrick.nelson@yahoo.com
23	melissa_cc cartermeli	Manager	melissa.carter@gmail.com
24	brian_mitic mitchellbri	Supervisor	brian.mitchell@yahoo.com
25	stephanie_perezstepf	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 philippstim	Supervisor	timothy.phillips@yahoo.com
29	elizabeth_evanseliza	Manager	elizabeth.evans@gmail.com
30	ryan_collii collinsryan	Supervisor	ryan.collins@yahoo.com

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

```
LINES TERMINATED BY '\n';
```

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

6	Jessica	shopper30	Jessica Gai	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.jac	12
13	Daniel	shopper10	Daniel Har	daniel.hari	13
14	Amanda	shopper12	Amanda Le	amanda.le	14
15	Joshua	shopper13	Joshua Scc	joshua.sco	15
16	Nicole	shopper14	Nicole Gre	nicole.gree	16
17	Justin	shopper15	Justin Bak	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.mitcl	21
22	Stephanie	shopper20	Stephanie	stephanie.	22
23	Jason	shopper21	Jason Rob	jason.robe	23
24	Katherine	shopper22	Katherine	katherine.t	24
25	Timothy	shopper23	Timothy Pl	timothy.pl	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]

```
1 • use warehouse_management_system_dbms_project;
2 • INSERT INTO ADMIN (Username, Password, Role, Email) VALUES
3 ('john_doe', 'password123', 'Manager', 'john.doe@gmail.com');
```

Automatic context help disabled. Use the toolbar manually get help for the current caret position or toggle automatic help.



#	Time	Action	Message	Duration / Fetch
1	11:38:51	use warehouse_management_system_dbms_project	0 row(s) affected	0.000 sec
2	11:38:56	INSERT INTO ADMIN (Username, Password, Role, Email) VALUES ('john_doe', 'password123', 'Manager', 'john.d...')	1 row(s) affected	0.031 sec

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

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

The screenshot shows the MySQL Workbench interface. In the SQL editor, a script is being run:

```
5 • SET FOREIGN_KEY_CHECKS = 0;
6 • TRUNCATE TABLE ADMIN;
7 • INSERT INTO ADMIN (Username, Password, Role, Email) VALUES
8 ('john_doe', 'password123', 'Manager', 'john.doe@gmail.com'),
9 ('jane_smith', 'adminpass', 'Supervisor', 'jane.smith@gmail.com'),
10 ('alice_lee', 'leeealice456', 'Manager', 'alice.lee@gmail.com'),
11 ('bob_jones', 'bobsecurepass', 'Supervisor', 'bob.jones@gmail.com'),
12 ('susan_clark', 'clarksecure123', 'Manager', 'susan.clark@gmail.com');
13
14
```

The output window shows the results of the executed statements:

#	Time	Action	Message	Duration / Fetch
1	11:44:23	SET FOREIGN_KEY_CHECKS = 0	0 row(s) affected	0.000 sec
2	11:44:23	TRUNCATE TABLE ADMIN	0 row(s) affected	0.047 sec
3	11:44:29	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, 'leeealice456', 'Manager', alice.lee@gmail.com), (bob_jones, 'bobsecurepass', 'Supervisor', bob.jones@gmail.com), (susan_clark, 'clarksecure123', 'Manager', susan.clark@gmail.com)	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.016 sec

Hence the execution time to insert each row with a single insert statement would be $5 \times 0.031 = 0.155$ 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', 'leeealice456', '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');
```

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

```
('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					
	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	lealice456	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								
	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						
	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 Clothinas	Matthew Hall	matthew.hall@mail.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: Wrap Cell Content:						
	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;`

The screenshot shows a MySQL query results grid titled "Result Grid". The query executed was "SELECT * FROM BorrowRequest;". The grid has columns: RequestID, RequestDate, BorrowStatus, ProductID, and ShopkeeperID. The data consists of 5 rows:

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 attributes. 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

The screenshot shows a MySQL query results grid titled "Result Grid". The query executed was "select * from LOCATION;". The grid has columns: locationId, ZipCode, Email, Street, ManagerName, and OpeningHours. The data consists of 3 rows:

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

```

	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;

```

	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 USER EXPERIENCE DESIGN

11.1.1 UX 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.

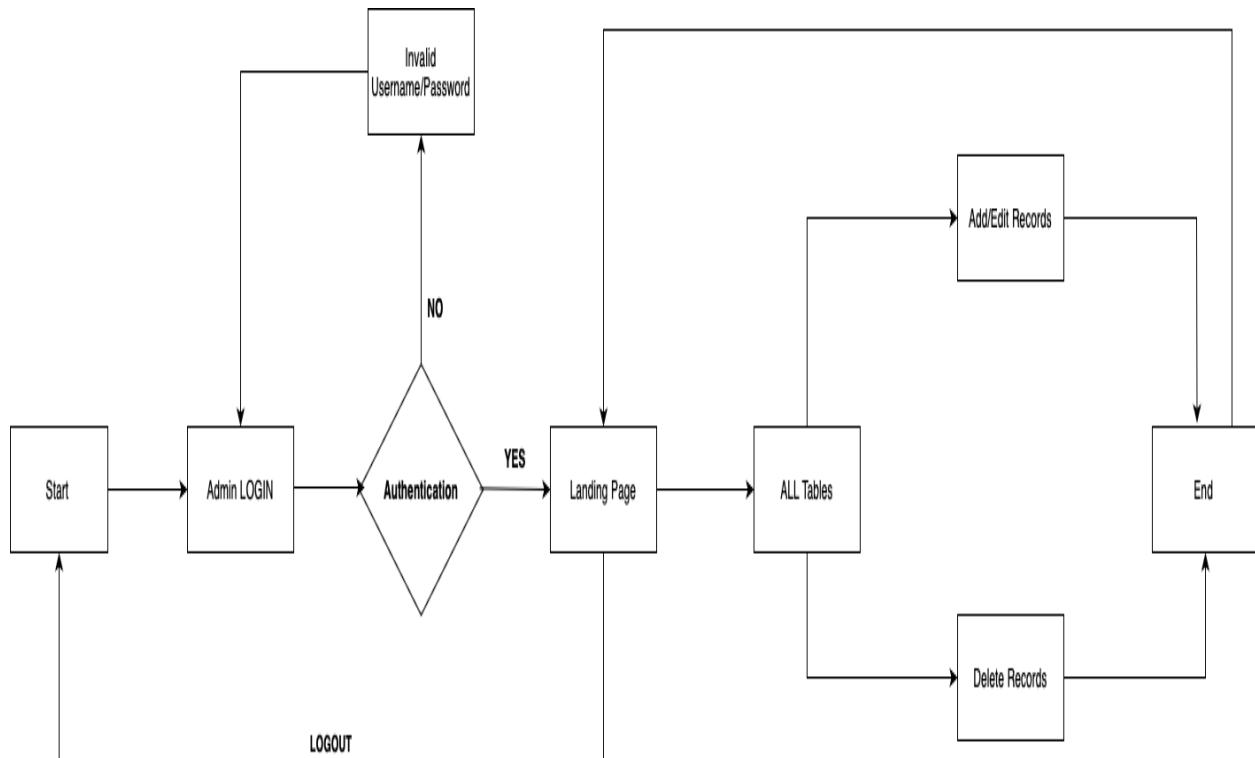


Figure 11.1.1 UX Admin POV

11.1.2 UX 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 their past transaction history
 - **Borrow Request Screen:** User can request the product from warehouse

5. Product Catalog 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.

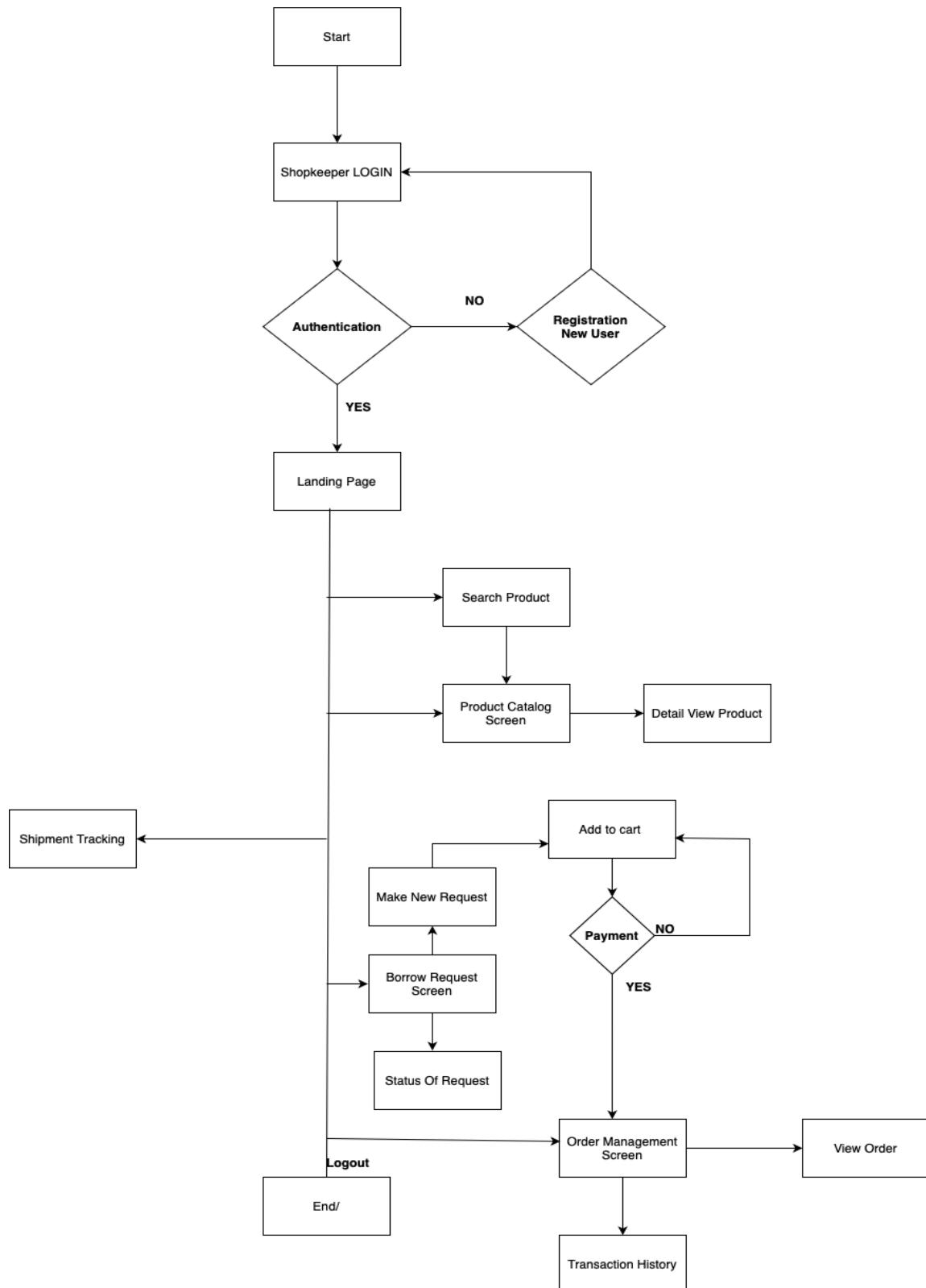


Figure 11.1.2 UX Shopkeeper POV

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. SECURITY CONTROL MEASURES IN DATABASE SYSTEM

Database security is crucial to protect sensitive information from unauthorized access, inference, and misuse. Implementing robust security control measures ensures data integrity, confidentiality, and availability. The primary security control measures include:

1. Access Control
2. Inference Control
3. Flow Control
4. Data Encryption

Below, we demonstrate how to implement these security measures in a MySQL database for the Warehouse Management System.

1. Access Control

Access Control restricts access to database resources based on user roles and privileges. By creating users with specific permissions, we can ensure that each user can only perform actions appropriate for their role.

Implementation Steps

- Create two users: an admin user (Sumanth) and a regular user (Kalyan).
- Assign different privileges to each user.
- Provide examples of tasks each user can and cannot perform.

Code and Explanation

- a. Create Two Users

```
-- Create an admin user  
CREATE USER 'Sumanth'@'localhost' IDENTIFIED BY 'password1';  
  
-- Create a regular user  
CREATE USER 'Kalyan'@'localhost' IDENTIFIED BY 'password';
```

- Sumanth: Will have full administrative privileges on the database.
- Kalyan: Will have limited privileges, such as read-only access to specific tables.

b. Assign Different Privileges

- Admin User Privileges

```
-- Grant All Privileges on the Entire Database  
  
GRANT SELECT, INSERT, UPDATE, CREATE ON  
`warehouse_management_system_dbms_project`.* TO  
'Sumanth'@'localhost';  
  
GRANT DELETE ON  
`warehouse_management_system_dbms_project`. `ADMIN` TO  
'Sumanth'@'localhost';  
  
GRANT DELETE ON  
`warehouse_management_system_dbms_project`. `CATEGORY`  
TO 'Sumanth'@'localhost';  
  
GRANT DELETE ON  
`warehouse_management_system_dbms_project`. `SHIPMENT`  
TO 'Sumanth'@'localhost';
```

```

GRANT DELETE ON
`warehouse_management_system_dbms_project`.`SUPPLIER`
TO 'Sumanth'@'localhost';

GRANT DELETE ON
`warehouse_management_system_dbms_project`.`TRANSACTION`
` TO 'Sumanth'@'localhost';

```

Sumanth now has full control over all objects in the Warehouse_Management_System_DBMS_Project database except delete on orders and products table.

13 • SHOW GRANTS FOR 'Sumanth'@'localhost';	
	Result Grid Filter Rows: Export: Wrap Cell Content:
Grants for Sumanth@localhost	
▶	GRANT USAGE ON *.* TO 'Sumanth'@'localhost'
	GRANT SELECT, INSERT, UPDATE, CREATE ON `warehouse_management_system_dbms_project`.* TO 'Sumanth'@'localhost'
	GRANT DELETE ON `warehouse_management_system_dbms_project`.`admin` TO 'Sumanth'@'localhost'
	GRANT DELETE ON `warehouse_management_system_dbms_project`.`category` TO 'Sumanth'@'localhost'
	GRANT DELETE ON `warehouse_management_system_dbms_project`.`shipment` TO 'Sumanth'@'localhost'
	GRANT DELETE ON `warehouse_management_system_dbms_project`.`supplier` TO 'Sumanth'@'localhost'
	GRANT DELETE ON `warehouse_management_system_dbms_project`.`transaction` TO 'Sumanth'@'localhost'

- Regular User Privileges

```

GRANT SELECT ON
Warehouse_Management_System_DBMS_Project.PRODUCT TO
'Kalyan`@`localhost';

GRANT SELECT ON
Warehouse_Management_System_DBMS_Project.CATEGORY TO
'Kalyan`@`localhost';

```

Kalyan can only perform SELECT operations on the PRODUCT and CATEGORY tables.

15 • SHOW GRANTS FOR 'Kalyan'@'localhost';			
Output			
Action Output			
#	Time	Action	Message
✓ 1	12:29:54	CREATE USER 'Kalyan'@'localhost' IDENTIFIED BY 'password'	0 row(s) affected
✓ 2	12:29:59	GRANT SELECT ON Warehouse_Management_System_DBMS_Project.'PRODUCT' TO ...	0 row(s) affected
✓ 3	12:30:07	GRANT SELECT ON Warehouse_Management_System_DBMS_Project.'CATEGORY' TO ...	0 row(s) affected
✓ 4	12:30:11	SHOW GRANTS FOR 'Kalyan'@'localhost'	3 row(s) returned

c. Examples of Accessible Tasks

- Admin User

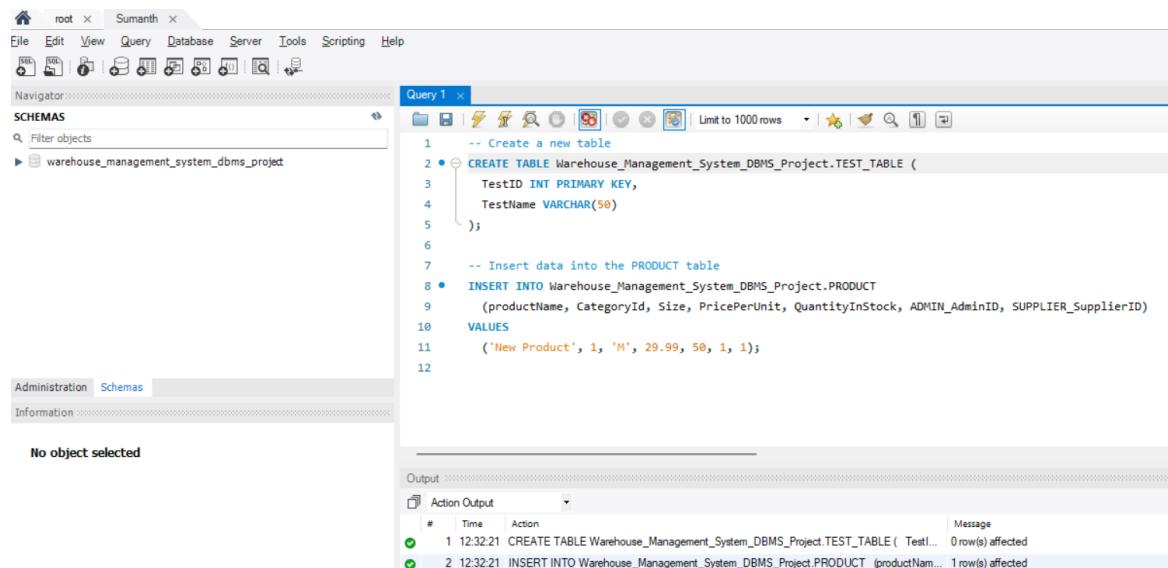
Sumanth can perform administrative tasks like creating tables, inserting data, and modifying schemas.

```
-- Create a new table
CREATE TABLE
Warehouse_Management_System_DBMS_Project.TEST_TABLE (
    TestID INT PRIMARY KEY,
    TestName VARCHAR(50)
);
```

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

```
-- Insert data into the PRODUCT table

INSERT INTO
Warehouse_Management_System_DBMS_Project.PRODUCT
(productName, CategoryId, Size, PricePerUnit,
QuantityInStock, ADMIN_AdminID, SUPPLIER_SupplierID)
VALUES
('New Product', 1, 'M', 29.99, 50, 1, 1);
```



- Regular User

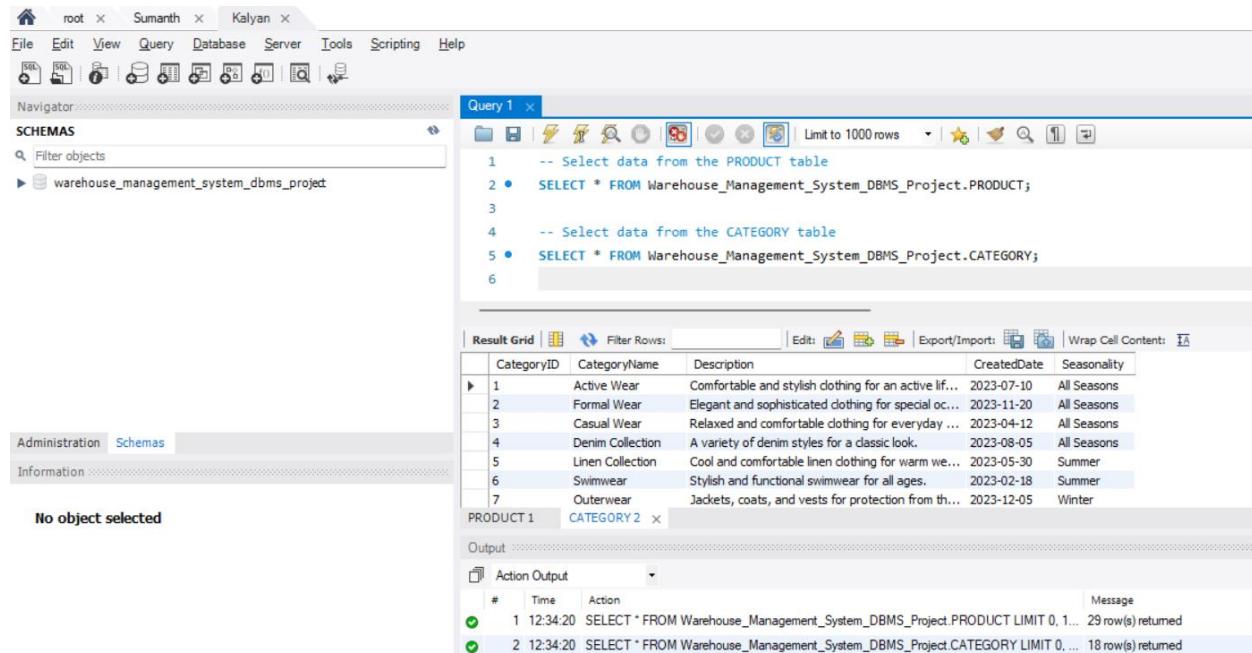
Kalyan can perform read-only operations on the permitted tables.

```
-- Select data from the PRODUCT table

SELECT * FROM
Warehouse_Management_System_DBMS_Project.PRODUCT;
```

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

```
-- Select data from the CATEGORY table  
  
SELECT * FROM  
Warehouse_Management_System_DBMS_Project.CATEGORY;
```



The screenshot shows the MySQL Workbench interface. The top menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. The left sidebar has a Navigator section with SCHEMAS and a Filter objects search bar. Below it, the Schemas list shows 'warehouse_management_system_dbms_project'. The main area is titled 'Query 1' and contains the following SQL code:

```
-- Select data from the PRODUCT table  
SELECT * FROM Warehouse_Management_System_DBMS_Project.PRODUCT;  
  
-- Select data from the CATEGORY table  
SELECT * FROM Warehouse_Management_System_DBMS_Project.CATEGORY;
```

Below the code is a 'Result Grid' table with columns: CategoryID, CategoryName, Description, CreatedDate, and Seasonality. The data is as follows:

CategoryID	CategoryName	Description	CreatedDate	Seasonality
1	Active Wear	Comfortable and stylish clothing for an active life...	2023-07-10	All Seasons
2	Formal Wear	Elegant and sophisticated clothing for special occasions...	2023-11-20	All Seasons
3	Casual Wear	Relaxed and comfortable clothing for everyday wear...	2023-04-12	All Seasons
4	Denim Collection	A variety of denim styles for a classic look.	2023-08-05	All Seasons
5	Linen Collection	Cool and comfortable linen clothing for warm weather...	2023-05-30	Summer
6	Swimwear	Stylish and functional swimwear for all ages.	2023-02-18	Summer
7	Outerwear	Jackets, coats, and vests for protection from the elements.	2023-12-05	Winter

At the bottom, the 'Output' tab shows the 'Action Output' log with two entries:

#	Time	Action	Message
1	12:34:20	SELECT * FROM Warehouse_Management_System_DBMS_Project.PRODUCT LIMIT 0, 1...	29 row(s) returned
2	12:34:20	SELECT * FROM Warehouse_Management_System_DBMS_Project.CATEGORY LIMIT 0, 1...	18 row(s) returned

d. Examples of Tasks Out of Their Privileges

- Admin User

Sumanth cannot access databases or tables they don't have privileges and cannot delete from product, orders table.

```
-- Attempt to select data from a different database  
  
SELECT * FROM sakila.actor;  
  
-- Error: Access denied for user 'Sumanth'@'localhost'  
to database
```

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

The screenshot shows the MySQL Workbench interface. In the top-left, there's a navigation pane with tabs for 'Administration' and 'Schemas'. The 'Schemas' tab is selected, showing a list with 'warehouse_management_system_dbms_project'. Below this is an 'Information' pane with the message 'No object selected'. The main area is a 'Query 1' editor containing the following SQL code:

```

-- Create a new table
CREATE TABLE Warehouse_Management_System_DBMS_Project.TEST_TABLE (
    TestID INT PRIMARY KEY,
    TestName VARCHAR(50)
);

-- Insert data into the PRODUCT table
INSERT INTO Warehouse_Management_System_DBMS_Project.PRODUCT
    (productName, CategoryId, Size, PricePerUnit, QuantityInStock, ADMIN_AdminID, SUPPLIER_SupplierID)
VALUES
    ('New Product', 1, 'M', 29.99, 50, 1, 1);

-- SELECT * FROM sakila.actor;

```

To the right of the editor is a 'SQLAdditions' panel with a note about automatic context being disabled. The bottom right contains 'Context Help' and 'Snippets' buttons.

Output:

#	Time	Action	Message	Duration /
1	12:32:21	CREATE TABLE Warehouse_Management_System_DBMS_Project.TEST_TABLE (Test... 0 row(s) affected		0.031 sec
2	12:32:21	INSERT INTO Warehouse_Management_System_DBMS_Project.PRODUCT (productName, CategoryId, Size, PricePerUnit, QuantityInStock, ADMIN_AdminID, SUPPLIER_SupplierID) VALUES ('New Product', 1, 'M', 29.99, 50, 1, 1); 1 row(s) affected		0.032 sec
3	12:35:30	SELECT * FROM sakila.actor LIMIT 0, 1000	Error Code: 1142. SELECT command denied to user 'Sumanth'@localhost for table 'actor'	0.000 sec

-- Admin should not be able to DELETE from PRODUCT or ORDER:

```
DELETE FROM PRODUCT WHERE productID = 1; -- Error:
Access denied
```

```
DELETE FROM `ORDER` WHERE OrderId = 1; -- Error:
Access denied
```

The screenshot shows the MySQL Workbench interface. The 'Schemas' tab is selected, showing 'warehouse_management_system_dbms_project'. The 'Information' pane says 'No object selected'. The 'Query 1' editor contains the following SQL code:

```

SELECT * FROM sakila.actor;
DELETE FROM Warehouse_Management_System_DBMS_Project.`PRODUCT` WHERE productID = 1; -- Error: Access denied
DELETE FROM Warehouse_Management_System_DBMS_Project.`ORDER` WHERE OrderId = 1; -- Error: Access denied

```

The 'Output' pane shows the results of these queries:

#	Time	Action	Message	Duration /
1	12:37:44	DELETE FROM Warehouse_Management_System_DBMS_Project.`PRODUCT` WHERE pr... Error Code: 1142. DELETE command denied to user 'Sumanth'@localhost for table 'product'		0.015 sec
2	12:37:44	DELETE FROM Warehouse_Management_System_DBMS_Project.`ORDER` WHERE Ord... Error Code: 1142. DELETE command denied to user 'Sumanth'@localhost for table 'order'		0.000 sec

- Regular User

Kalyan cannot perform write operations or access unauthorized tables.

```
-- Attempt to insert data into the PRODUCT table
INSERT INTO
Warehouse_Management_System_DBMS_Project.PRODUCT
(productName, CategoryId, Size, PricePerUnit,
QuantityInStock, ADMIN_AdminID, SUPPLIER_SupplierID)
VALUES
('Unauthorized Product', 2, 'L', 39.99, 30, 1, 2);
-- Error: INSERT command denied to user
'Kalyan'@'localhost' for table 'PRODUCT'

-- Attempt to select data from the ADMIN table
SELECT * FROM
Warehouse_Management_System_DBMS_Project.ADMIN;
-- Error: SELECT command denied to user
'Kalyan'@'localhost' for table 'ADMIN'
```

The screenshot shows the MySQL Workbench interface. In the top-left corner, there are tabs for 'root', 'Sumanth', and 'Kalyan'. The main area is a 'Query 1' editor containing the following SQL code:

```

7 -- Attempt to insert data into the PRODUCT table
8 • INSERT INTO Warehouse_Management_System_DBMS_Project.PRODUCT
9   (productName, CategoryId, Size, PricePerUnit, QuantityInStock, ADMIN_AdminID, SUPPLIER_SupplierID)
10  VALUES
11    ('Unauthorized Product', 2, 'L', 39.99, 30, 1, 2);
12  -- Error: INSERT command denied to user 'Kalyan'@'localhost' for table 'PRODUCT'
13
14  -- Attempt to select data from the ADMIN table
15 • SELECT * FROM Warehouse_Management_System_DBMS_Project.ADMIN;
16  -- Error: SELECT command denied to user 'Kalyan'@'localhost' for table 'ADMIN'

```

In the bottom right corner of the Query Editor, there is a note: "Automatic disabled. Use manually current connection toggle at".

Below the Query Editor is an 'Output' pane titled 'Action Output' with the following log entries:

#	Time	Action	Message
1	12:34:20	SELECT * FROM Warehouse_Management_System_DBMS_Project.PRODUCT LIMIT 0, 29 row(s) returned	
2	12:34:20	SELECT * FROM Warehouse_Management_System_DBMS_Project.CATEGORY LIMIT 0, 18 row(s) returned	
3	12:38:57	INSERT INTO Warehouse_Management_System_DBMS_Project.PRODUCT (productName, CategoryId, Size, PricePerUnit, QuantityInStock, ADMIN_AdminID, SUPPLIER_SupplierID) VALUES ('Unauthorized Product', 2, 'L', 39.99, 30, 1, 2);	Error Code: 1142: INSERT command denied to user 'Kalyan'@'localhost' for table 'product'
4	12:38:57	SELECT * FROM Warehouse_Management_System_DBMS_Project.ADMIN LIMIT 0, 1000	Error Code: 1142: SELECT command denied to user 'Kalyan'@'localhost' for table 'admin'

2. Inference Control

Description

Inference Control prevents users from deducing sensitive information from the data they can access. By providing summarized data through views and restricting access to detailed data, we can limit the potential for unauthorized inference.

Implementation Steps

- Create a summarized view that aggregates data without exposing details.
- Grant access to the view to a specific group of users (e.g., administration).
- Verify access by logging in as a user with permissions to the view.

Code and Explanation

a. Create a Summarized View

Since we already created a view of `vw_SupplierPerformance` we are gonna use that here.

b. Grant Access to the View

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

```
-- Admin Already Has Access to Detailed Data Of  
Supplier, Products
```

```
-- Grant Regular User Access to the Summarized View
```

```
GRANT SELECT ON  
warehouse_management_system_dbms_project.vw_SupplierPer  
formance TO 'Kalyan'@'localhost';
```

```
5 12:39:58 GRANT SELECT ON warehouse_management_system_dbms_project.vw_SupplierPerform... 0 row(s) affected 0.016 sec
```

c. Verify Access

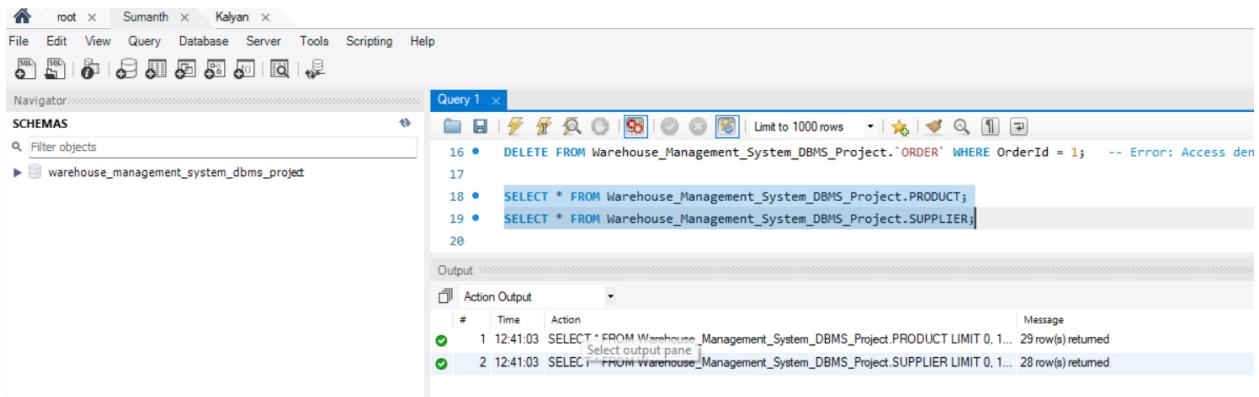
- Admin User

```
-- Login as Sumanth
```

```
-- As admin_user (Detailed Data):
```

```
SELECT * FROM  
Warehouse_Management_System_DBMS_Project.PRODUCT;  
  
SELECT * FROM  
Warehouse_Management_System_DBMS_Project.SUPPLIER;
```

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys



The screenshot shows the Oracle SQL Developer interface. The top menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. The title bar indicates the session is for 'Kalyan'. The Navigator pane shows the schema 'warehouse_management_system_dbms_project'. The Query 1 pane contains the following SQL code:

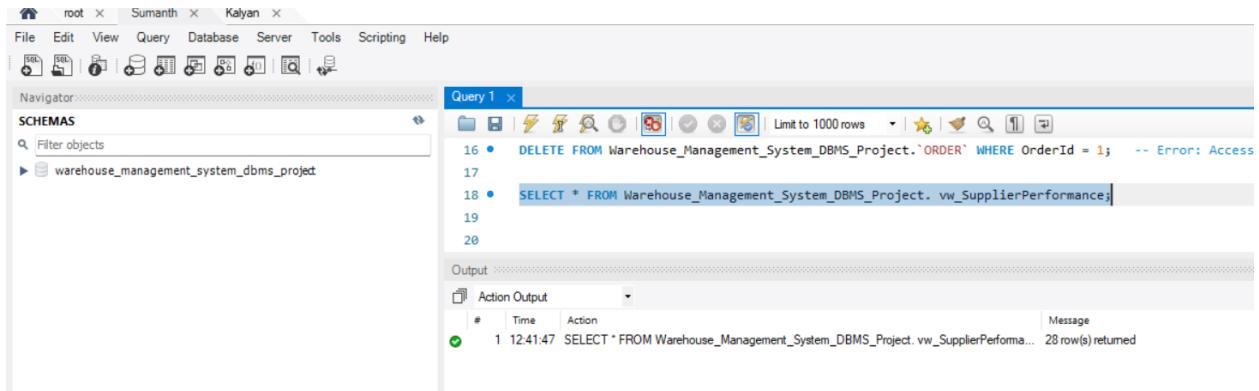
```
16 • DELETE FROM Warehouse_Management_System_DBMS_Project.ORDER WHERE OrderId = 1; -- Error: Access den  
17  
18 • SELECT * FROM Warehouse_Management_System_DBMS_Project.PRODUCT;  
19 • SELECT * FROM Warehouse_Management_System_DBMS_Project.SUPPLIER;  
20
```

The Output pane shows the results of the last two queries:

#	Time	Action	Message
1	12:41:03	SELECT * FROM Warehouse_Management_System_DBMS_Project.PRODUCT LIMIT 0, 1...	29 row(s) returned
2	12:41:03	SELECT * FROM Warehouse_Management_System_DBMS_Project.SUPPLIER LIMIT 0, 1...	28 row(s) returned

-- Query the summarized view

```
SELECT * FROM Warehouse_Management_System_DBMS_Project.  
vw_SupplierPerformance;
```



The screenshot shows the Oracle SQL Developer interface. The top menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. The title bar indicates the session is for 'Kalyan'. The Navigator pane shows the schema 'warehouse_management_system_dbms_project'. The Query 1 pane contains the following SQL code:

```
16 • DELETE FROM Warehouse_Management_System_DBMS_Project.ORDER WHERE OrderId = 1; -- Error: Access den  
17  
18 • SELECT * FROM Warehouse_Management_System_DBMS_Project.vw_SupplierPerformance;  
19  
20
```

The Output pane shows the results of the last query:

#	Time	Action	Message
1	12:41:47	SELECT * FROM Warehouse_Management_System_DBMS_Project.vw_SupplierPerformance	28 row(s) returned

- Regular User

-- Login as Kalyan

-- As Kalyan (Only Summarized Data):

```
SELECT * FROM
Warehouse_Management_System_DBMS_Project.vw_SupplierPerformance;
```

The screenshot shows the MySQL Workbench interface. The top menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. The 'View' tab is selected. The 'Navigator' pane on the left shows the database structure, including the schema 'warehouse_management_system_dbms_project'. The main area is titled 'Query 1' and contains the SQL query: 'SELECT * FROM Warehouse_Management_System_DBMS_Project.vw_SupplierPerformance;'. Below the query is a 'Result Grid' table with the following data:

SupplierID	SupplierName	ContactPerson	Email	TotalProductsSupplied	TotalInventorySupplied
1	Textile Co	Laura Anderson	laura.anderson@yahoo.com	2	100
2	Cloth Suppliers Inc	Kevin White	kevin.white@gmail.com	1	30
3	Global Fabrics	Angela Green	angela.green@yahoo.com	1	100
4	Elite Clothings	Matthew Hall	matthew.hall@gmail.com	1	40
5	Fashion Fabrics	Jessica Kino	jessica.kino@yahoo.com	1	20

The bottom pane shows the 'Output' log with one entry: '1 12:42:40 SELECT * FROM Warehouse_Management_System_DBMS_Project.vw_SupplierPerformance... 28 row(s) returned'.

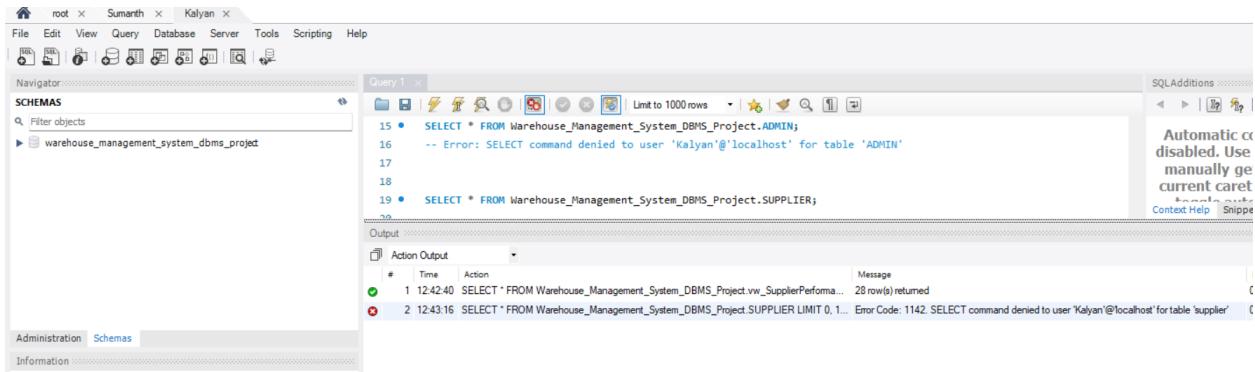
d. Restrict Access to Detailed Data

Ensure that Kalyan cannot access the underlying tables containing detailed information.

-- Attempt as Kalyan to access detailed data of supplier (should fail):

```
SELECT * FROM SUPPLIER;
```

```
-- Error: Access denied-- Error: SELECT command denied
to user 'Kalyan'@'localhost' for table 'SUPPLIER'
```



3. Flow Control

Description

Flow Control regulates the flow of information within the database to prevent unauthorized data leakage. By securely handling authentication and data processing, we can ensure that sensitive information doesn't reach unauthorized users.

Implementation Steps

- Alter the login table to store passwords in hashed form using SHA2.
- Insert instances into the login table with hashed passwords.
- Create a stored procedure that authenticates users by comparing hashed passwords.

Code and Explanation

a. Alter the Login Table to Store Hashed Passwords

Since SHA2-256 produces a 64-character hexadecimal string, we need to adjust the Password column in the ADMIN table.

-- Modify the Password field to accommodate SHA2 hash

```
ALTER TABLE
Warehouse_Management_System_DBMS_Project.ADMIN
```

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

```
MODIFY COLUMN Password CHAR(64) NOT NULL;
```

#	Time	Action	Message	Duration / Fetch
1	12:44:04	ALTER TABLE Warehouse_Management_System_DBMS_Project.ADMIN MODIFY COLUM...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.031 sec

b. Insert 3 Instances into the Login Table

Insert admin users with their passwords hashed using SHA2.

```
-- Insert admin users with hashed passwords

INSERT INTO
Warehouse_Management_System_DBMS_Project.ADMIN
(Username, Password, Role, Email)

VALUES

('admin1', SHA2('password1', 256), 'Manager',
'admin1@example.com'),

('admin2', SHA2('password2', 256), 'Supervisor',
'admin2@example.com'),

('admin3', SHA2('password3', 256), 'Manager',
'admin3@example.com');
```

27	•	INSERT INTO Warehouse_Management_System_DBMS_Project.ADMIN (Username, Password, Role, Email)	
28		VALUES	
29		('admin1', SHA2('password1', 256), 'Manager', 'admin1@example.com'),	
30		('admin2', SHA2('password2', 256), 'Supervisor', 'admin2@example.com'),	
31		('admin3', SHA2('password3', 256), 'Manager', 'admin3@example.com');	
32			
33			
34			

#	Time	Action	Message
1	12:44:04	ALTER TABLE Warehouse_Management_System_DBMS_Project.ADMIN MODIFY COLUM...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
2	12:44:40	INSERT INTO Warehouse_Management_System_DBMS_Project.ADMIN (Username, Pass...	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0

c. Create a Stored Procedure for Authentication

```
DELIMITER $$
```

```
CREATE PROCEDURE
Warehouse_Management_System_DBMS_Project.AuthenticateUs
er (
    IN inputUsername VARCHAR(50),
    IN inputPassword VARCHAR(50),
    OUT resultMessage VARCHAR(50)
)

BEGIN
    DECLARE storedPassword CHAR(64);
    DECLARE hashedInputPassword CHAR(64);

    -- Hash the input password using SHA2-256
    SET hashedInputPassword = SHA2(inputPassword, 256);

    -- Retrieve the stored hashed password for the
    -- given username
    SELECT Password INTO storedPassword
    FROM Warehouse_Management_System_DBMS_Project.ADMIN
    WHERE Username = inputUsername;
```

```

-- Check if the username exists
IF storedPassword IS NOT NULL THEN
    -- Compare the hashed passwords
    IF storedPassword = hashedInputPassword THEN
        SET resultMessage = 'Access granted';
    ELSE
        SET resultMessage = 'Access denied';
    END IF;
ELSE
    SET resultMessage = 'Access denied';
END IF;

END $$

DELIMITER ;

```

The screenshot shows the MySQL Workbench interface with the following details:

- Code Area:** Displays the SQL code for the stored procedure.
- Output Area:** Shows the execution log with three entries:

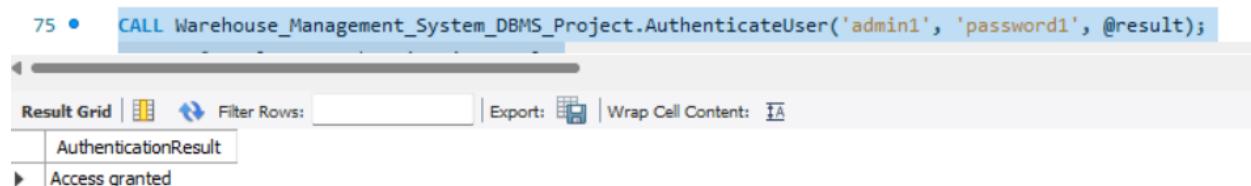
#	Action	Time	Message	Duration / Fetch
1	ALTER TABLE	12:44:04	Warehouse_Management_System_DBMS_Project.ADMIN MODIFY COLUM...	0.031 sec
2	INSERT INTO	12:44:40	Warehouse_Management_System_DBMS_Project.ADMIN (Username, Pass...)	0.016 sec
3	CREATE PROCEDURE	12:45:13	Warehouse_Management_System_DBMS_Project.AuthenticateUs...	0.015 sec
- Right Panel:** Contains a tooltip about automatic context help being disabled and instructions to use the toolbar for help.

d. Test the Stored Procedure

- Successful Authentication

-- Test with correct username and password

```
CALL  
Warehouse_Management_System_DBMS_Project.AuthenticateUser('admin1', 'password1', @result);  
  
SELECT @result AS AuthenticationResult;
```



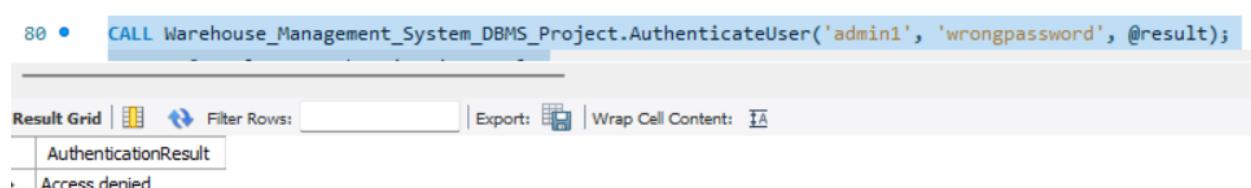
A screenshot of a database query results window. The SQL command is shown at the top: `CALL Warehouse_Management_System_DBMS_Project.AuthenticateUser('admin1', 'password1', @result);`. Below the command is a result grid with one row. The first column is labeled "AuthenticationResult" and contains the value "Access granted". There are buttons for "Result Grid", "Filter Rows", "Export", and "Wrap Cell Content".

AuthenticationResult
Access granted

- Failed Authentication (Wrong Password)

-- Test with correct username but wrong password

```
CALL  
Warehouse_Management_System_DBMS_Project.AuthenticateUser('admin1', 'wrongpassword', @result);  
  
SELECT @result AS AuthenticationResult;
```



A screenshot of a database query results window. The SQL command is shown at the top: `CALL Warehouse_Management_System_DBMS_Project.AuthenticateUser('admin1', 'wrongpassword', @result);`. Below the command is a result grid with one row. The first column is labeled "AuthenticationResult" and contains the value "Access denied". There are buttons for "Result Grid", "Filter Rows", "Export", and "Wrap Cell Content".

AuthenticationResult
Access denied

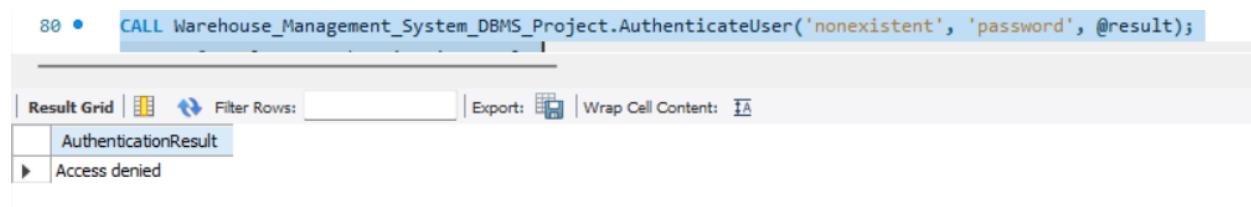
- Failed Authentication (Non-existent Username)

```
-- Test with non-existent username
```

```
CALL
```

```
Warehouse_Management_System_DBMS_Project.AuthenticateUser('nonexistent', 'password', @result);
```

```
SELECT @result AS AuthenticationResult;
```



The screenshot shows a database query results grid. At the top, there is a status bar with the number '80' and a message 'CALL Warehouse_Management_System_DBMS_Project.AuthenticateUser('nonexistent', 'password', @result);'. Below the status bar is a toolbar with icons for Result Grid, Filter Rows, Export, and Wrap Cell Content. The main area is a table with one row. The first column is labeled 'AuthenticationResult' and contains the value 'Access denied'.

AuthenticationResult
Access denied

4. Data Encryption

Description

Data Encryption protects sensitive data by converting it into an unreadable format for unauthorized users. Encryption ensures that even if data is accessed without authorization, it remains unintelligible without the decryption key.

Implementation Steps

- Use the AES encryption function to encrypt data in a table.
- Decrypt the encrypted data to retrieve the original information when needed.

Code and Explanation

a. Encrypt a Record Using AES Function

Let's encrypt the Email column in the ADMIN table.

i. Modify the Email Column to Store Encrypted Data

```
-- Modify the Email column to VARBINARY to store  
encrypted data
```

```
ALTER TABLE  
Warehouse_Management_System_DBMS_Project.ADMIN  
MODIFY COLUMN Email VARBINARY(255) NOT NULL;
```

```
✓ 10 12:48:49 ALTER TABLE Warehouse_Management_System_DBMS_Project.ADMIN MODIFY COL... 32 row(s) affected Records: 32 Duplicates: 0 Warnings: 0
```

ii. Encrypt the Email Addresses

```
-- Define a secret key for encryption
```

```
SET @encryption_key = 'mysecretkey123';
```

```
-- Encrypt the Email column
```

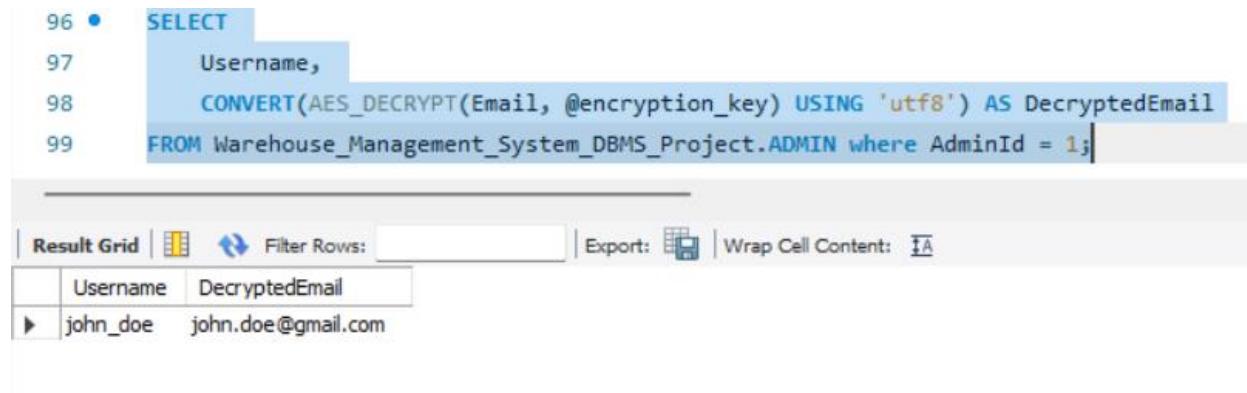
```
UPDATE Warehouse_Management_System_DBMS_Project.ADMIN  
SET Email = AES_ENCRYPT(Email, @encryption_key) where  
AdminId = 1;
```

b. Decrypt the Encrypted Results

To retrieve the original email addresses, we'll use AES_DECRYPT.

```
-- Decrypt the Email column to retrieve original emails  
SELECT
```

```
Username,  
        CONVERT(AES_DECRYPT(Email, @encryption_key) USING  
'utf8') AS DecryptedEmail  
FROM Warehouse_Management_System_DBMS_Project.ADMIN;
```



The screenshot shows a database query results grid. The query is:

```
96 •  SELECT  
97      Username,  
98      CONVERT(AES_DECRYPT(Email, @encryption_key) USING 'utf8') AS DecryptedEmail  
99      FROM Warehouse_Management_System_DBMS_Project.ADMIN where AdminId = 1;
```

The results grid has two columns: "Username" and "DecryptedEmail". There is one row of data:

Username	DecryptedEmail
john_doe	john.doe@gmail.com

These measures work together to protect the database from unauthorized access, inference, and data breaches, ensuring data integrity and confidentiality.

13.PASSWORD MANAGEMENT

The change_pass stored procedure is designed to enhance the security and manageability of user passwords in the ADMIN or SHOPKEEPER tables. It takes three input parameters: username, password, and new_password. The procedure first validates the provided credentials by comparing the hashed value of the input password with the stored password in the database. If the credentials are valid, the procedure updates the password with the hashed version of the new_password and notifies the user of a successful update. If the credentials are invalid, it informs the user that the provided password is incorrect. This approach ensures secure handling of user authentication and password management.

```

DELIMITER $$

CREATE PROCEDURE change_pass(
    IN username VARCHAR(50),
    IN password VARCHAR(50),
    IN new_password VARCHAR(50)
)

BEGIN
    DECLARE KnowsOldPassword INT;

    -- Check if the username and hashed password match
    -- a record in the database
    SELECT COUNT(*) INTO KnowsOldPassword
    FROM ADMIN
    WHERE Username = username AND Password =
        SHA2(password, 256);

```

```
IF KnowsOldPassword > 0 THEN
    -- Update password with the new hashed password
    UPDATE ADMIN
    SET Password = SHA2(new_password, 256)
    WHERE Username = username;
    SELECT 'The password has been updated
successfully.' AS Result;
ELSE
    -- Inform user about invalid password
    SELECT 'The provided password is not valid.' AS
Result;
END IF;
END$$

DELIMITER ;
```

a) Successful Password Update

● **Input Parameters:**

- username: admin1
- password: password1
- new_password: password23

● **Output:**

```
275 • CALL change_pass('admin1', 'password1', 'password23');
276
100% 14:275 |
```

Result Grid Filter Rows: Search Export:

Result
The password has been updated successfully.

b) Wrong Password Update

- **Input Parameters:**

- username: admin1
- password: wrongpass
- new_password: password23

- **Output:**

```
275 • CALL change_pass('admin1', 'wrongpass', 'password23');
276
100% 38:275 |
```

Result Grid Filter Rows: Search Export:

Result
The provided password is not valid.

14. STORAGE ENGINE

a) BLACKHOLE Engine:

The username field is set as the primary key. We used the BLACKHOLE storage engine for this table. The BLACKHOLE engine accepts data but does not store it; any data inserted into the table is immediately discarded. This was an opportunity to learn about a storage engine that behaves differently from the traditional ones. The BLACKHOLE engine is useful for scenarios where data storage is not required, such as logging or replication. Data inserted into a BLACKHOLE table cannot be retrieved with a SELECT statement. Creating a table with the BLACKHOLE engine is similar to any other table creation, with the only difference being the engine specification.

```
CREATE TABLE
Warehouse_Management_System_DBMS_Project.Trush (
    FailedAttempts TINYINT,
    SuccessfulAttempts TINYINT,
    username VARCHAR(50),
    PRIMARY KEY (username)
) ENGINE = BLACKHOLE;
```

```

1 •   use `warehouse_management_system_dbms_project`;
2 •   CREATE TABLE Warehouse_Management_System_DBMS_Project.Trush (
3     FailedAttempts TINYINT,
4     SuccessfulAttempts TINYINT,
5     username VARCHAR(50),
6     PRIMARY KEY (username)
7   ) ENGINE = BLACKHOLE;

```

Output:

Action Output			
#	Time	Action	Message
1	13:26:11	use `warehouse_management_system_dbms_project`	0 row(s) affected
2	13:28:35	CREATE TABLE Warehouse_Management_System_DBMS_Project.Trush (F...	0 row(s) affected

We inserted three records into the Trush table to simulate tracking of access attempts for different users. Despite successful insertions, we anticipated that the data would not be retrievable due to the BLACKHOLE engine's nature. The BLACKHOLE engine accepts insertions without errors. No data is actually stored; SELECT queries return zero rows. This behavior confirms that BLACKHOLE is not suitable for tables where data retrieval is necessary.

```

-- Insert data into Trush table
INSERT INTO
Warehouse_Management_System_DBMS_Project.Trush
(FailedAttempts, SuccessfulAttempts, username)
VALUES (2, 5, 'admin1');

INSERT INTO
Warehouse_Management_System_DBMS_Project.Trush
(FailedAttempts, SuccessfulAttempts, username)
VALUES (1, 3, 'admin2');

```

```
INSERT INTO
Warehouse_Management_System_DBMS_Project.Trush
(FailedAttempts, SuccessfulAttempts, username)
VALUES (0, 4, 'admin3');
```

```
-- Attempt to select data from Trush table
SELECT * FROM
Warehouse_Management_System_DBMS_Project.Trush;
```

```
2 •  SELECT * FROM Warehouse_Management_System_DBMS_Project.Trush;
3
4
5
```

The screenshot shows the MySQL Workbench interface with a result grid. The grid has three columns: FailedAttempts, SuccessfulAttempts, and username. A single row is present with all values set to NULL. Below the grid, there is an output panel labeled 'Action Output'.

	FailedAttempts	SuccessfulAttempts	username
*	NULL	NULL	NULL

The SELECT query returns zero rows because the BLACKHOLE engine does not store data.

We attempted to perform a JOIN operation between the Inventory and Trush tables using the shared column primaryUser from Inventory and username from Trush. The goal was to see if we could combine data from both tables based on the common user. The JOIN operation executes without syntax errors. The resultset is empty because the Trush table does not store any data due to the BLACKHOLE engine. This demonstrates that joining with a BLACKHOLE table yields no data,

highlighting the importance of choosing appropriate storage engines based on data retrieval needs.

- Attempt to join Inventory and Trush tables

SELECT

```
i.itemID,  
i.item,  
i.Availability,  
i.primaryUser,  
t.FailedAttempts,  
t.SuccessfulAttempts  
FROM Warehouse_Management_System_DBMS_Project.Inventory  
i  
JOIN Warehouse_Management_System_DBMS_Project.Trush t  
ON i.primaryUser = t.username;
```

```

1 •   SELECT
2       i.itemID,
3       i.item,
4       i.Availability,
5       i.primaryUser,
6       t.FailedAttempts,
7       t.SuccessfulAttempts
8   FROM Warehouse_Management_System_DBMS_Project.Inventory i
9   JOIN Warehouse_Management_System_DBMS_Project.Trush t
10      ON i.primaryUser = t.username;
11

```

Result Grid					
itemID	item	Availability	primaryUser	FailedAttempts	SuccessfulAttempts

The query returns zero rows because the Trush table contains no data. Acts as a data sink; data inserted is immediately discarded. Not suitable for tables where data storage and retrieval are required. Useful for logging, replication, or testing scenarios where data persistence is unnecessary.

b) MyISAM Engine:

We used the MyISAM storage engine for this table. MyISAM is known for fast read operations but does not support transactions or foreign key constraints. This task allowed us to understand the characteristics and limitations of the MyISAM engine. The MyISAM engine is suitable for applications that require quick read access and do not need transaction support. It does not enforce foreign key constraints, so referential integrity must be maintained manually or through application logic. Data is stored on disk and can be retrieved using standard SELECT queries.

```

CREATE TABLE
Warehouse_Management_System_DBMS_Project.Inventory (
    itemID INT,
    item VARCHAR(255),
    Availability INT,
    primaryUser VARCHAR(50),
    PRIMARY KEY (itemID)
) ENGINE = MyISAM;

```

```

• CREATE TABLE Warehouse_Management_System_DBMS_Project.Inventory (
    itemID INT,
    item VARCHAR(255),
    Availability INT,
    primaryUser VARCHAR(50),
    PRIMARY KEY (itemID)
) ENGINE = MyISAM;

```

	Action	Time	Message
1	use `warehouse_management_system_dbms_project`	13:26:11	0 row(s) affected
2	CREATE TABLE Warehouse_Management_System_DBMS_Project.Trush (F...	13:28:35	0 row(s) affected

We inserted three records into the Inventory table to represent different items managed by different users. This allowed me to observe how the MyISAM engine stores and retrieves data. Data inserted into the Inventory table is stored and can be retrieved. The MyISAM engine behaves as expected for standard data storage and retrieval operations. Fast read access is a key feature of MyISAM.

-- Insert data into Inventory table

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

```
INSERT INTO
Warehouse_Management_System_DBMS_Project.Inventory
(itemID, item, Availability, primaryUser)
VALUES (1, 'Product A', 100, 'admin1');

INSERT INTO
Warehouse_Management_System_DBMS_Project.Inventory
(itemID, item, Availability, primaryUser)
VALUES (2, 'Product B', 50, 'admin2');

INSERT INTO
Warehouse_Management_System_DBMS_Project.Inventory
(itemID, item, Availability, primaryUser)
VALUES (3, 'Product C', 75, 'admin3');

-- Select data from Inventory table
SELECT * FROM
Warehouse_Management_System_DBMS_Project.Inventory;
```

1 • **SELECT * FROM Warehouse_Management_System_DBMS_Project.Inventory;**
2

The screenshot shows a 'Result Grid' window with the following data:

	itemID	item	Availability	primaryUser
▶	1	Product A	100	admin1
	2	Product B	50	admin2
	3	Product C	75	admin3
*	HULL	HULL	HULL	HULL

Stores data on disk and allows for fast read operations. Does not support transactions or enforce foreign key constraints. Suitable for read-heavy applications where data integrity can be managed externally.

15. DATABASE INTEGRITY

In this phase of the project, We implemented a database transaction to maintain database integrity when removing a username from the login table (ADMIN table). The transaction ensures that when a user is deleted, all items in the Inventory table that have the same primaryUser are updated to another existing user. This way, both actions (deleting the user and updating the inventory records) are performed atomically—either both succeed, or neither does—ensuring there are no orphaned references and maintaining referential integrity.

Changed the storage engine of the Inventory table from MyISAM to InnoDB to support transactions and foreign key constraints.

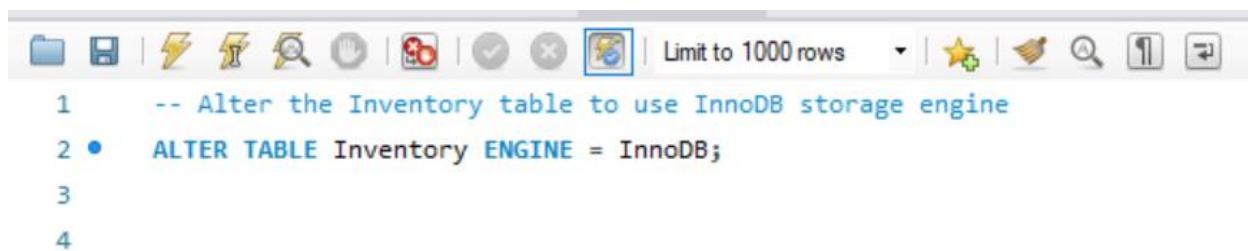
Implemented the Transaction: Began a transaction using START TRANSACTION. Deleted the user from the ADMIN table. Updated the Inventory table, setting primaryUser to another existing user where it matched the deleted username. Committed the transaction with COMMIT.

By using a transaction, I ensured that if any part of the process fails, the entire transaction is rolled back, and the database remains consistent. This protects the database integrity by preventing partial updates that could lead to data inconsistencies.

1. Modify the Inventory Table to Use InnoDB

```
-- Alter the Inventory table to use InnoDB storage  
engine
```

```
ALTER TABLE Inventory ENGINE = InnoDB;
```



1 -- Alter the Inventory table to use InnoDB storage engine
2 • ALTER TABLE Inventory ENGINE = InnoDB;
3
4

2. Ensure Usernames Exist in the ADMIN Table

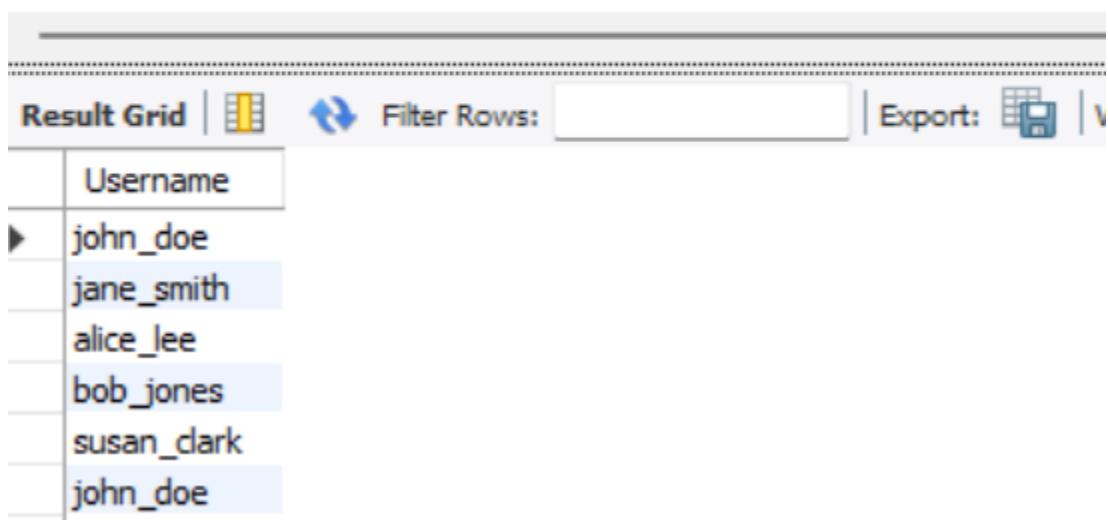
For this transaction, let's assume we have the following users in the ADMIN table:

-- Existing users in the ADMIN table

```
SELECT Username FROM ADMIN;
```



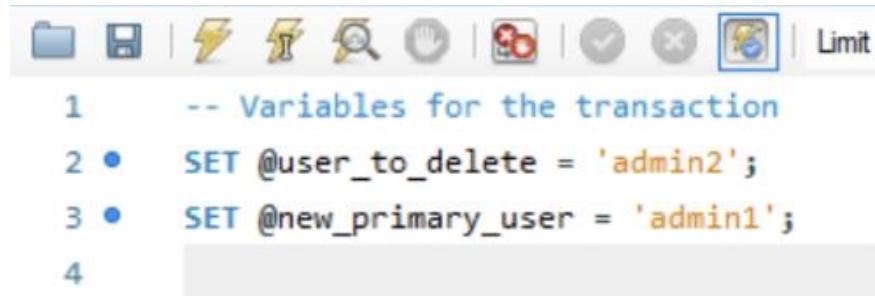
1 • SELECT Username FROM ADMIN;



Username
john_doe
jane_smith
alice_lee
bob_jones
susan_dark
john_doe

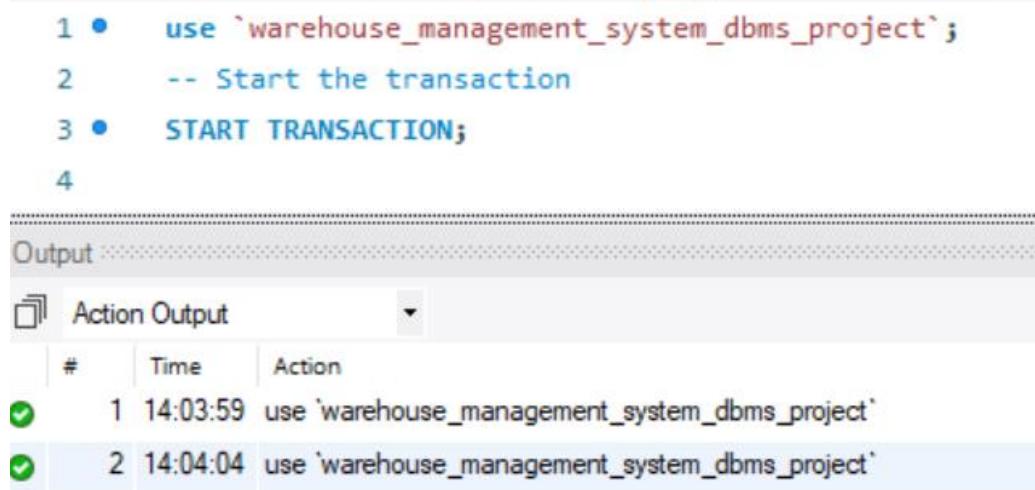
3. Transaction to Delete a User and Update Inventory

```
-- Variables for the transaction  
SET @user_to_delete = 'admin2';  
SET @new_primary_user = 'admin1';
```



```
1 -- Variables for the transaction  
2 • SET @user_to_delete = 'admin2';  
3 • SET @new_primary_user = 'admin1';  
4
```

START TRANSACTION;



```
1 • use `warehouse_management_system_dbms_project`;  
2 -- Start the transaction  
3 • START TRANSACTION;  
4
```

Output

Action Output		
#	Time	Action
1	14:03:59	use `warehouse_management_system_dbms_project`
2	14:04:04	use `warehouse_management_system_dbms_project`

```
-- Delete the user from the ADMIN table  
DELETE FROM ADMIN WHERE Username = @john_deo;
```

```
1      -- Delete the user from the ADMIN table
2 •  DELETE FROM ADMIN WHERE Username = @john_deo;
3
4
```

Output

Action Output

#	Time	Action
1	14:07:22	DELETE FROM ADMIN WHERE Username = @user_to_delete
2	14:08:15	DELETE FROM ADMIN WHERE Username = @john_deo

-- Update the Inventory table

UPDATE Inventory

SET primaryUser = @sagarroy

WHERE primaryUser = @jane_smith;

```
1      -- Update the Inventory table
2      UPDATE Inventory
3      SET primaryUser = @sagarroy
4      WHERE primaryUser = @jane_smith;
```

Output

Action Output

#	Time	Action
1	14:07:22	DELETE FROM ADMIN WHERE Username = @user_to_delete
2	14:08:15	DELETE FROM ADMIN WHERE Username = @john_deo
3	14:09:41	UPDATE Inventory SET primaryUser = @sagarroy WHERE primaryUser = @jane_smith

-- Commit the transaction

COMMIT;

```
1      -- Commit the transaction
2      COMMIT;
3
4
```

Output		
Action Output		
#	Time	Action
1	14:10:43	COMMIT

4. Explanation

- **Variables:**
 - @user_to_delete: The username to be removed from the ADMIN table.
 - @new_primary_user: The existing username to assign as the new primaryUser in the Inventory table.
- **Transaction Steps:**
 - **START TRANSACTION:** Begins a new transaction.
 - **DELETE:** Removes the specified user from the ADMIN table.
 - **UPDATE:** Updates all records in the Inventory table where primaryUser matches the deleted username, setting it to the new primary user.
 - **COMMIT:** Commits the transaction, making all changes permanent.
- **Integrity Protection:**
 - If the DELETE or UPDATE operation fails for any reason, the transaction can be rolled back, preventing partial updates.
 - This ensures that there are no records in Inventory referencing a non-existent user, maintaining referential integrity.

5. Verifying the Transaction

Before Transaction:

-- Check the Inventory table before the transaction

```
SELECT * FROM Inventory;
```

```
1 • SELECT * FROM Inventory;  
2
```

The screenshot shows a database result grid with the following data:

	itemID	item	Availability	primaryUser
▶	1	Product A	100	admin1
	2	Product B	50	admin2
	3	Product C	75	admin3
*	HULL	HULL	HULL	HULL

After Transaction:

```
-- Check the Inventory table after the transaction
```

```
SELECT * FROM Inventory;
```

The screenshot shows a MySQL Workbench interface. At the top, there are several icons: a folder, a table, a lightning bolt, a trophy, a magnifying glass, a circular progress bar, a red circle with a minus sign, a checkmark, a close button, and a refresh icon. To the right of these is a dropdown menu labeled "Limit to 1". Below the toolbar, a query editor window is open with the following content:

```
1 • select * from inventory;  
2
```

Below the query editor is a "Result Grid" window. It has tabs for "Result Grid" (which is selected), "Table", and "CSV". There are also buttons for "Filter Rows:" and "Edit:" with a pencil icon. The result grid displays the following data:

	itemID	item	Availability	primaryUser
▶	1	Product A	100	admin1
*	NULL	NULL	NULL	NULL

```
-- Verify the user has been deleted from ADMIN table  
SELECT * FROM ADMIN;
```

```

1      -- Verify the user has been deleted from ADMIN table
2 •  SELECT * FROM ADMIN;
3
4

```

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	leelalice456	Manager	alice.lee@gmail.com
	4	bob_jones	bobsecurepass	Supervisor	bob.jones@gmail.com
	5	susan_dark	darksecure123	Manager	susan.dark@gmail.com
	6	john_doe	password123	Manager	john.doe@gmail.com
	7	jane_smith	adminpass	Supervisor	jane.smith@gmail.com
	8	alice_lee	leelalice456	Manager	alice.lee@gmail.com
	9	bob_jones	bobsecurepass	Supervisor	bob.jones@gmail.com
*	10	susan_dark	darksecure123	Manager	susan.dark@gmail.com
	NULL	NULL	NULL	NULL	NULL

How the Transaction Protects Database Integrity

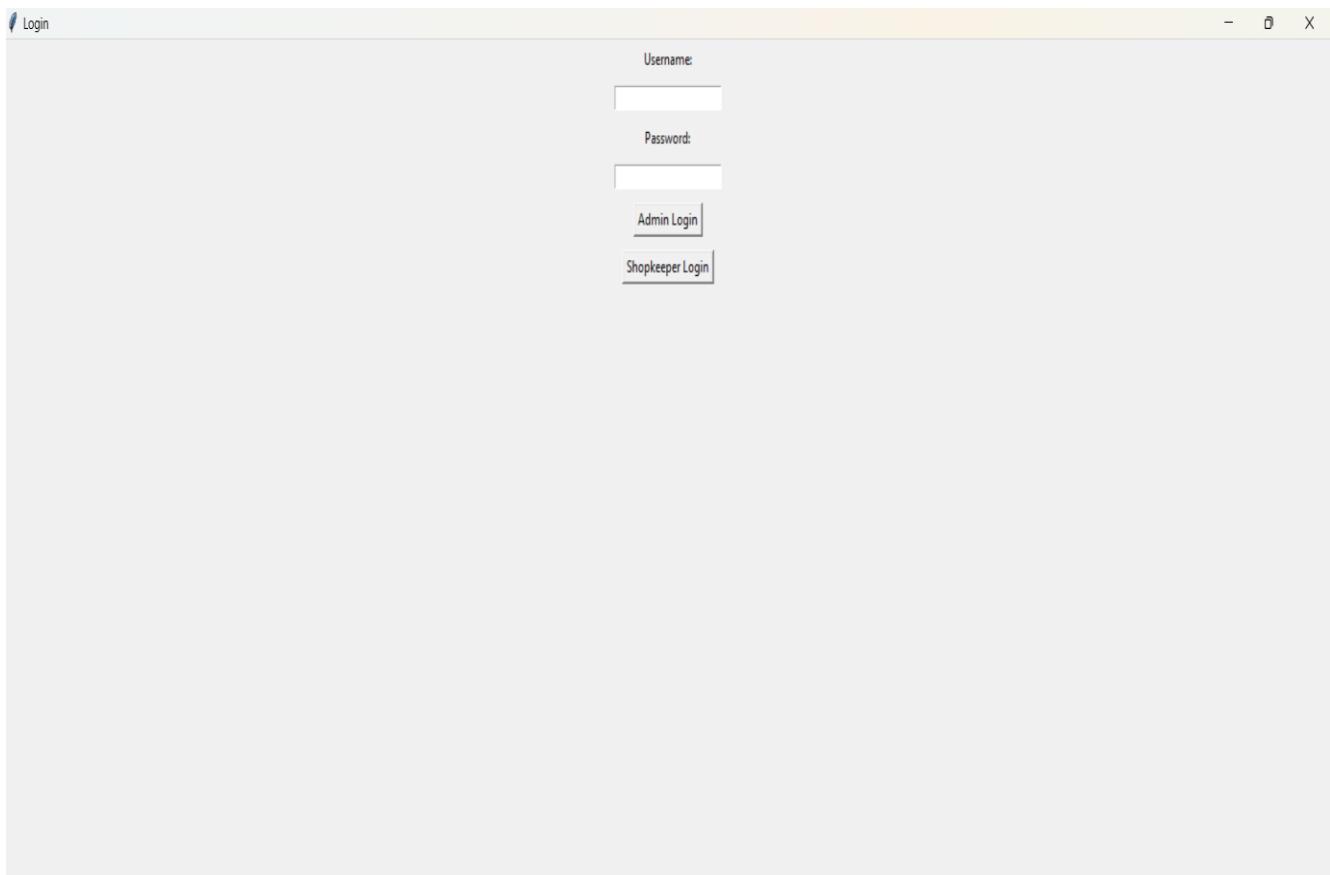
- **Atomicity:** The transaction ensures that the deletion of the user and the update of the Inventory records are treated as a single atomic operation. If any part fails, no changes are applied.
- **Consistency:** The database moves from one consistent state to another. By updating the Inventory records, we prevent references to a non-existent user, maintaining referential integrity.
- **Isolation:** While the transaction is in progress, other operations cannot interfere, ensuring data integrity during the process.
- **Durability:** Once the transaction is committed, the changes are permanent, even in the case of a system failure.

By carefully coordinating the deletion and update within a transaction, we prevent issues such as orphaned records or inconsistent data states, thus protecting the integrity of the database. Implementing the transaction as described ensures that any deletion of a user from the ADMIN table does not leave behind inconsistent data in the Inventory table. By updating the primaryUser field in the Inventory table within the same transaction, we maintain data integrity and prevent

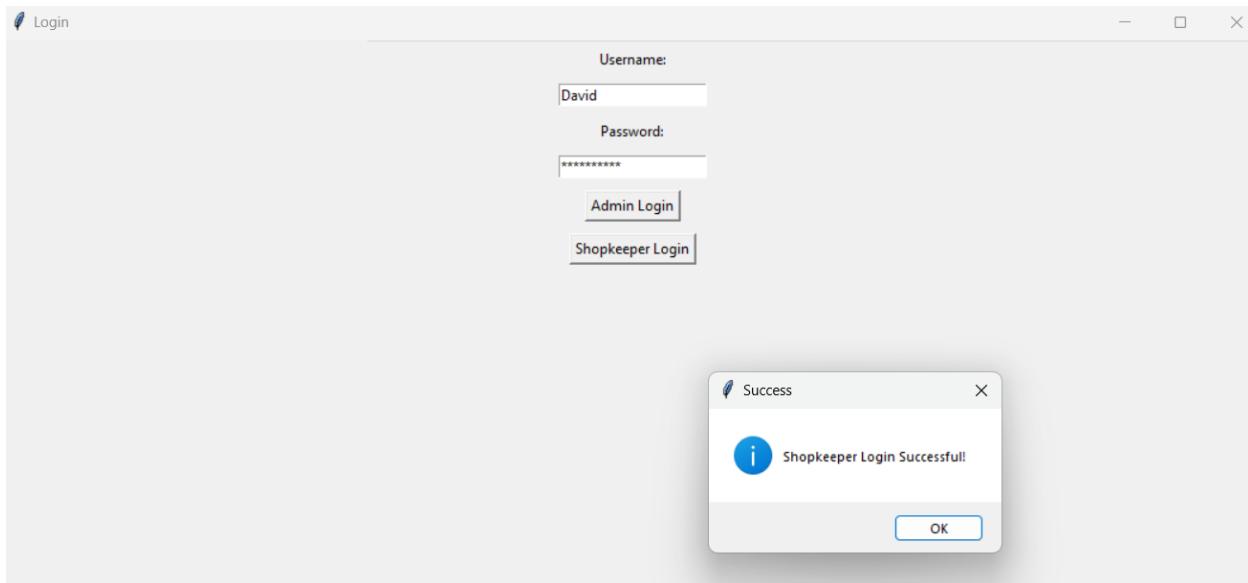
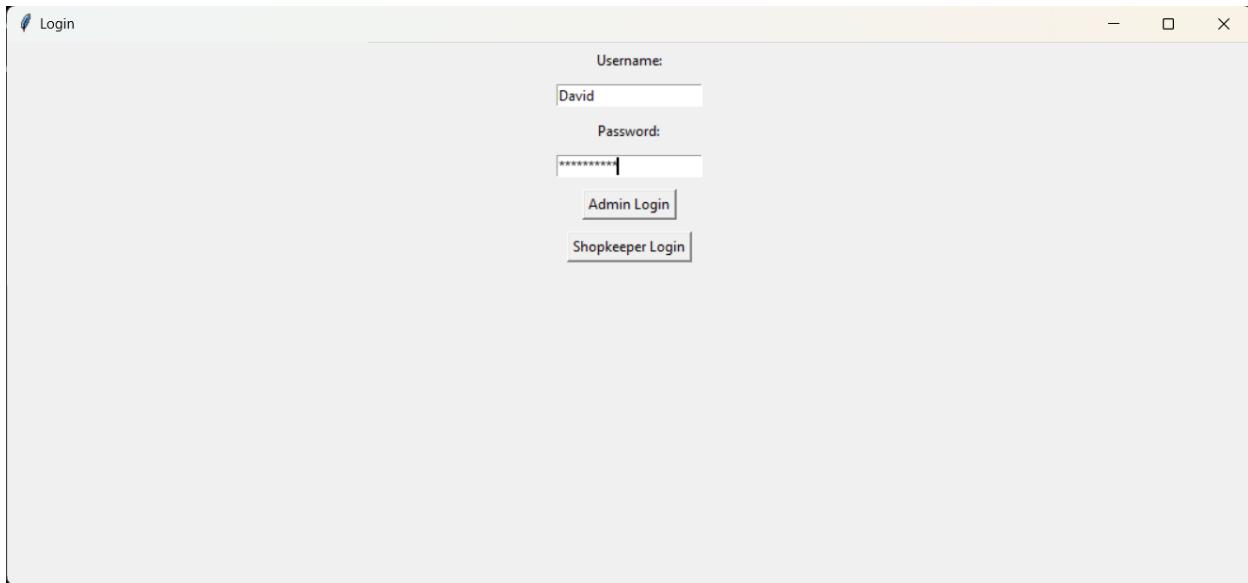
references to deleted users. This approach adheres to the ACID properties of transactions, providing a reliable and consistent database state.

16. GUI

- Login Window



- Shopkeeper Login



- Shopkeeper Dashboard

The screenshot shows the Shopkeeper Dashboard interface. At the top, there's a search bar labeled "Search Product:" with "Search" and "Show All" buttons. Below the search bar is a table titled "View Products" showing a list of items with columns for Product ID, Name, Size, Price/Unit, and Stock.

Product ID	Name	Size	Price/Unit	Stock
31	Cotton Shirt	L	19.99	60
32	Wooden Jacket	M	49.99	34
33	Silk Scarf	One Size	14.99	98
34	Denim Jeans	XL	39.99	40
35	Leather Gloves	M	24.99	20
36	Cotton Dress	S	29.99	60
37	Woolen Sweater	L	39.99	25
38	Silk Blouse	One Size	19.99	80
39	Denim Skirt	M	24.99	55
40	Leather Belt	One Size	12.99	120

Below the products section is a "Place Order" form with fields for "Product ID:" and "Quantity:", and a "Place Order" button.

At the bottom, there's a section titled "Your Orders" showing a list of orders with columns for Order ID, Order Date, Quantity, Status, and Amount.

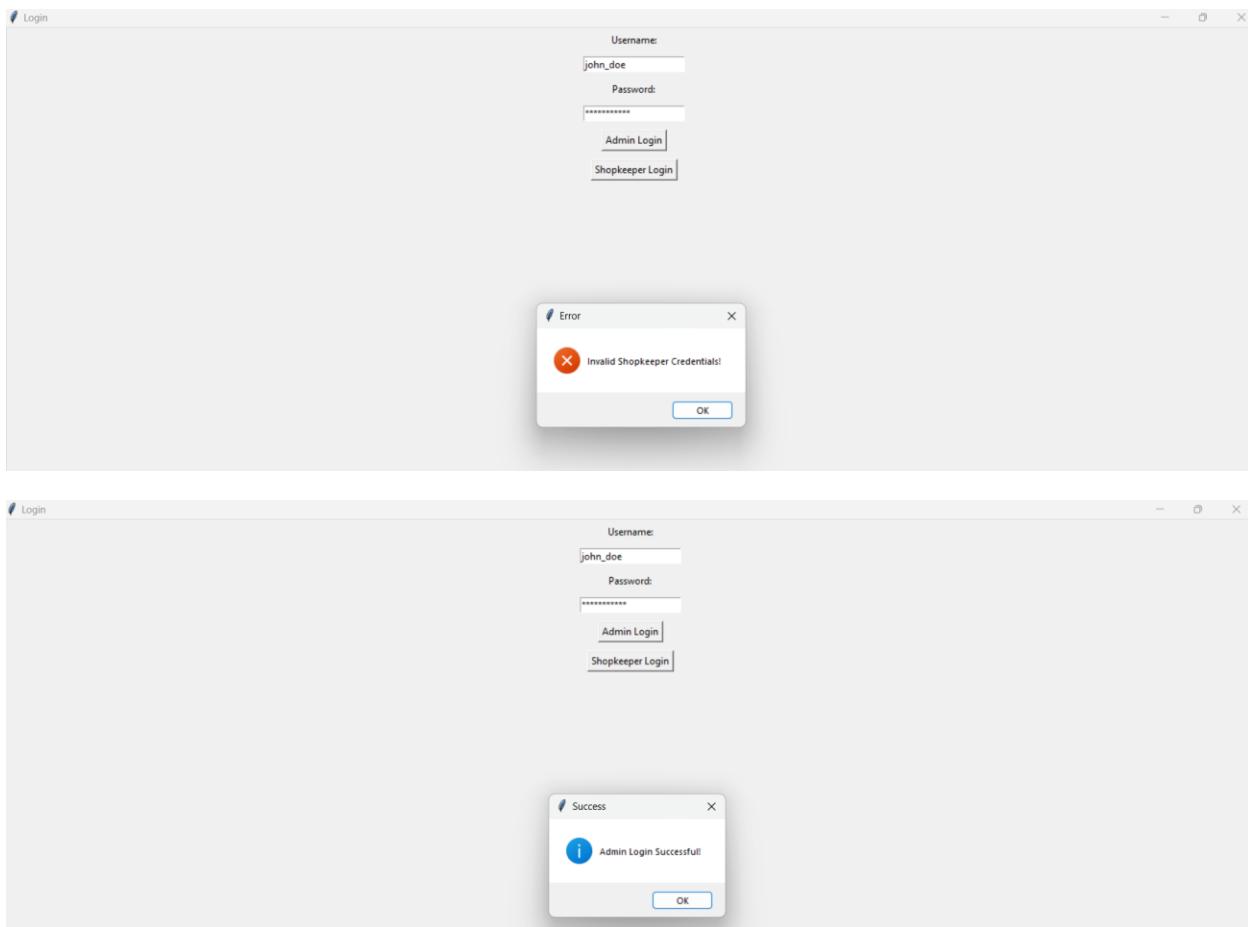
Order ID	Order Date	Quantity	Status	Amount
1	2024-09-01	10	Shipped	199.90
6	2024-12-02	2	Shipped	39.98
7	2024-12-02	1	Shipped	39.99
8	2024-12-02	2	Shipped	39.98
9	2024-12-02	2	Shipped	39.98
10	2024-12-02	2	Shipped	39.98
11	2024-12-02	1	Shipped	49.99
12	2024-12-03	2	Pending	99.98
13	2024-12-03	2	Shipped	29.98
14	2024-12-03	4	Pending	59.96

[\[Refresh Orders.\]](#)

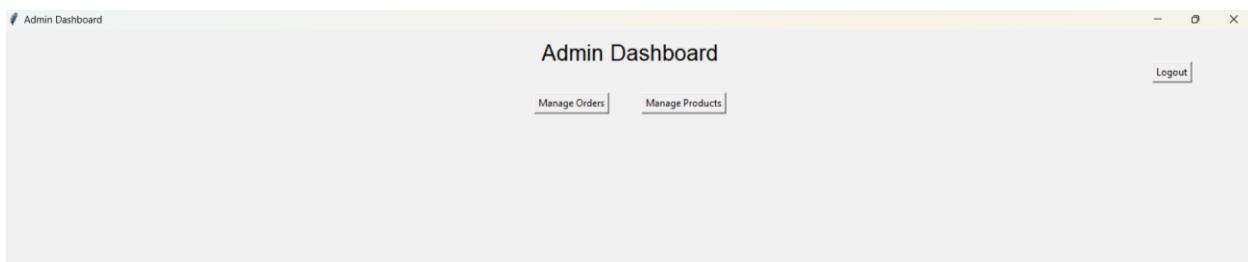
- Admin Login

The screenshot shows the Admin Login page. It features a logo icon and the word "Login". Below the logo is a "Username:" label with an input field containing "john_doe". Underneath is a "Password:" label with an input field showing masked text "*****". At the bottom are two buttons: "Admin Login" and "Shopkeeper Login".

MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys



● Admin Dashboard



MSCS_542L_256_24F: Project Progress Report: Phase10 TechBoys

- Admin Dashboard (Manage Products)

The screenshot shows the Admin Dashboard with the Manage Products tab selected. At the top, there are input fields for Product Name, Category ID, Size, Price Per Unit, and Quantity in Stock, each with a clear button. Below these are buttons for Add Product, Update Product, and Clear Fields. A table lists 10 products with columns for Product ID, Name, Category ID, Size, Price Per Unit, and Quantity in Stock. At the bottom are buttons for View Products and Logout.

Product ID	Name	Category ID	Size	Price Per Unit	Quantity in Stock
31	Cotton Shirt	1	L	19.99	60
32	Wooden Jacket	2	M	49.99	34
33	Silk Scarf	5	One Size	14.99	98
34	Denim Jeans	1	XL	39.99	40
35	Leather Gloves	2	M	24.99	20
36	Cotton Dress	1	S	29.99	60
37	Woolen Sweater	2	L	39.99	25
38	Silk Blouse	5	One Size	19.99	80
39	Denim Skirt	1	M	24.99	55
40	Leather Belt	5	One Size	12.99	120

- Admin Dashboard (Manage Orders)

The screenshot shows the Admin Dashboard with the Manage Orders tab selected. At the top, there are input fields for Order ID, Order Date, Shopkeeper ID, Quantity, and Amount, each with a clear button. Below these are buttons for View Pending Orders and Approve Order. A table lists pending orders with columns for Order ID, Order Date, Shopkeeper ID, Quantity, and Amount.

Order ID	Order Date	Shopkeeper ID	Quantity	Amount

The screenshot shows the Admin Dashboard with the Manage Orders tab selected. At the top, there are input fields for Order ID, Order Date, Shopkeeper ID, Quantity, and Amount, each with a clear button. Below these are buttons for View Pending Orders and Approve Order. A table lists completed orders with columns for Order ID, Order Date, Shopkeeper ID, Quantity, and Amount.

Order ID	Order Date	Shopkeeper ID	Quantity	Amount
12	2024-12-03	1	2	99.98
14	2024-12-03	1	4	59.96

17. REFERENCES

- [1] <https://www.zoho.com/us/inventory/warehouse-inventory-management-software/>
- [2] <https://www.fishbowlinventory.com/try/warehouse-management-google>
- [3] <https://www.koerber-supplychain-software.com/en>
- [4] <https://creately.com/diagram/example/ig4z3xv61/warehouse-management-system-er-diagram>
- [5] <https://www.geeksforgeeks.org/how-to-design-er-diagrams-for-supply-chain-management/>
- [6] <https://app.diagrams.net>
- [7] <https://www.sqltutorial.org/sql-data-types/>
- [8] <https://dev.mysql.com/doc/refman/8.0/en/string-type-syntax.html>
- [9] <https://dev.mysql.com/doc/refman/8.0/en/char.html>
- [10] <https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html>
- [11] <https://dev.mysql.com/doc/refman/8.0/en/blob.html>
- [12] <https://dev.mysql.com/doc/refman/8.0/en/enum.html>
- [13] <https://dev.mysql.com/doc/refman/8.0/en/set.html>
- [14] <https://learn.microsoft.com/en-us/sql/relational-databases/json/json-data-sql-server?view=sql-server-ver16>
- [15] <https://dev.mysql.com/doc/refman/8.0/en/column-indexes.html>
- [16] <https://dev.mysql.com/doc/refman/8.0/en/optimization-indexes.html>
- [17] <https://dev.mysql.com/doc/refman/8.4/en/insert-optimization.html>