

Warehouse Management System

Database Management System

MSCS 542L-256-24F

TechBoys



Marist College
School of Computer Science and Mathematics

Submitted To:
Dr. Reza Sadeghi

Oct 1, 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	7
7.2 ENHANCED ENTITY RELATIONSHIP MODEL (EER MODEL)	12
8. PRESENTATION SUBJECT	21
9. DATABASE DEVELOPMENT	38
9.1 DESCRIPTION:	38
9.2 CREATE STATEMENTS	47
10. LOADING DATA AND PERFORMANCE ENHANCEMENT	56
10.1 HANDLING KEY CONSTRAINTS:	56
10.2 IMPORTING DATA:	60
10.3 OPTIMIZATION:	63
11. REFERENCES	75

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

1. DESCRIPTION OF TEAM MEMBERS

Sumanth Kumar Katapally:

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

Abhijeet Cherungottil:

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

Sagar Shankaran:

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

2. INTRODUCTION

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

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

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

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

3. PROJECT OBJECTIVE

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

1. Admin Management:

- Secure login for the admin with the ability to change credentials.
- Add and remove shopkeepers by creating or deleting their usernames and passwords.
- Manage the warehouse inventory by adding, editing, and deleting items, including details such as item type, storage time, ID, name, quantities, price, and storage location.
- Review and manage borrowing requests, with the ability to accept or reject them.

2. Shopkeeper Features:

- Search for items in the warehouse based on various attributes like ID, name, and producer.
- Save a list of favorite items for quick reference.
- Request to borrow or purchase items for specific periods.
- View the history of borrowed items.

3. User Experience:

- Provide a welcoming page and a clear menu of functions on all pages.
- Display reports in a tabular format, showing organized lists of requested items.
- Include an exit function that thanks users for using the software.
- Implement warnings for specific actions, such as duplicate item IDs, exceeding borrowing limits, or null search results.

4. Security:

- Protect user information with optional encryption of passwords.

4. REVIEW

5.1 Zoho Inventory [1]:

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

5.2 Fishbowl Inventory [2]:

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

5.3 Korber Warehouse Management Systems [3]:

- Integrates with other supply chain solutions, such as yard management and transportation, and ERP systems.
- Is flexible and adaptable, with configuration functionality that reduces the need for extensive customization and development.
- Is available as a software-as-a-service (SaaS), which includes provisioning, maintenance, and support.
- Has over 99% inventory accuracy and 80-85% less inventory loss.

5. MERITS

1. Add a notification to alert the user to order more items when the count reaches 'n'
 - WMS proactively alerts users (admin or shopkeepers) when the inventory level of a particular item falls below a specified threshold (n), ensuring that the stock levels are replenished before they run out.
 - In older or manual systems, stock monitoring is typically done manually. Users might only realize an item is out of stock when it's too late, leading to delays in fulfilling orders.
2. Tracks the location of items within the warehouse, including specific aisles, racks, and bins.
 - WMS offers detailed location tracking for each item in the warehouse, down to the aisle, rack, and bin level, ensuring that every item can be quickly located.
 - In older systems or warehouses, item locations may be tracked in a general sense, such as by section or area, or even on paper. This can lead to inefficiencies in locating items.
3. Ensuring old stock is cleared before new stock is added.
 - Many traditional systems don't enforce stock rotation, meaning newer stock can be placed over older stock, increasing the risk of product damage
4. Manages items that are out of stock and ensures they are prioritized when they become available.
 - Out-of-stock items may not be tracked effectively in less advanced systems, which can result in delays in fulfilling backorders or not promptly notifying users when the items become available
5. Adding sustainability tracking.
 - Less advanced WMSs rarely integrate sustainability tracking. Most inventory systems focus purely on operational aspects like stock levels, ignoring environmental or sustainability impacts.

6. GITHUB REPOSITORY

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

7. ENTITY RELATIONSHIP MODEL (ER MODEL)

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

Entities	Attributes
ADMIN	AdminID, Username, Password, Role, Email
SHOPKEEPER	ShopkeeperID, Username, Password, FullName, Email
PRODUCT	productid, productName, CategoryId, Size, PricePerUnit, QuantityInStock
WAREHOUSE	WarehouseID, LocationId, MaxCapacity, CurrentCapacity, AvailableSpace
BORROWREQUEST	RequestID ,RequestDate,BorrowStatus,ProductID ,ShopkeeperID
CATEGORY	CategoryID ,CategoryName,Description,Createddate,Seasonality
SUPPLIER	SupplierID ,SupplierName,ContactPerson,Email,SuppliedCloths
ORDER	orderId, orderDate, ShopkeeperID, Orderstatus, Amount
LOCATION	locationId, Address, PhoneNumber, Manager, OpeningHours, TotalCapacity
TRANSACTION	TransactionID, OrderID, TransactionDate, PaymentMethod, Amount, DiscountApplied, FinalAmount, Invoice
STOCKAUDIT	AuditID, WarehouseID, AuditDate, AuditorName, Discrepancies
SHIPMENT	ShipmentID, OrderID, ShipmentDate, ShipmentStatus, TrackingNumber

7.1 IMPLEMENTATION OF ER DIAGRAM

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

- Rectangles - specifying entity types
- Ellipses - specifying attributes
- Rhomboids - specifying relationship types
- Lines - connections between entities and attributes, or entities and relationship types.

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

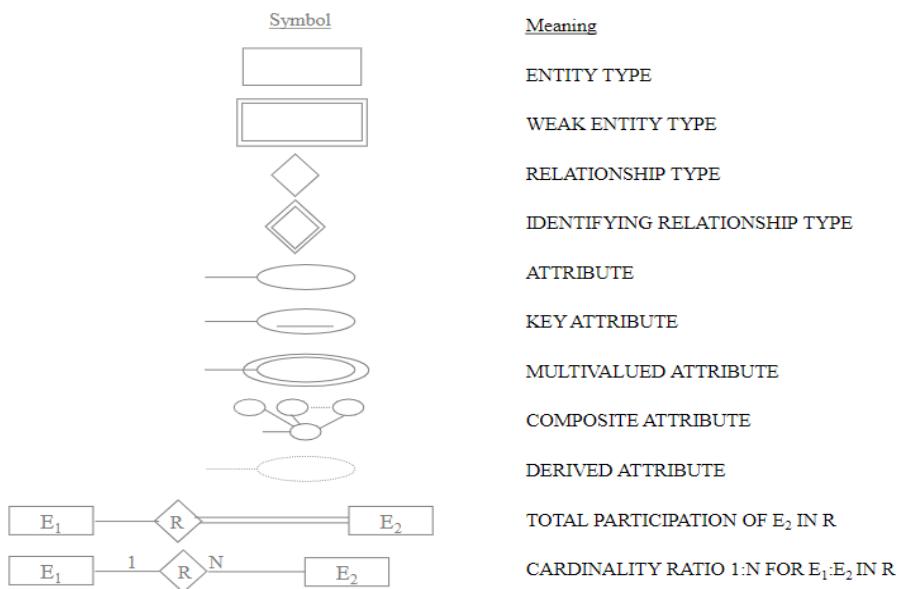


Figure 7. 1 Relationship of Entity Relationship Model

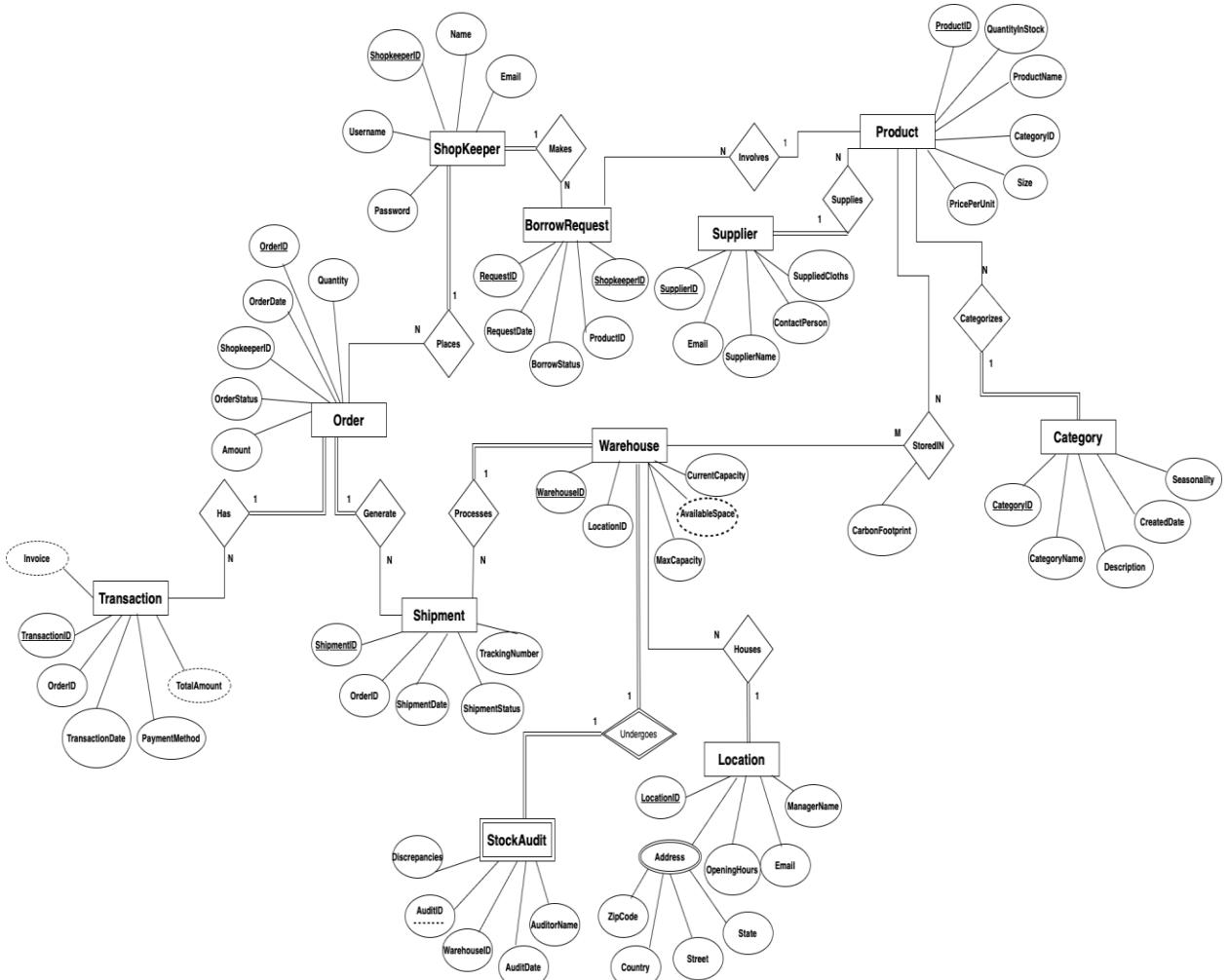


Figure 7.2 External ER SHOPKEEPER POV [6]

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

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

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

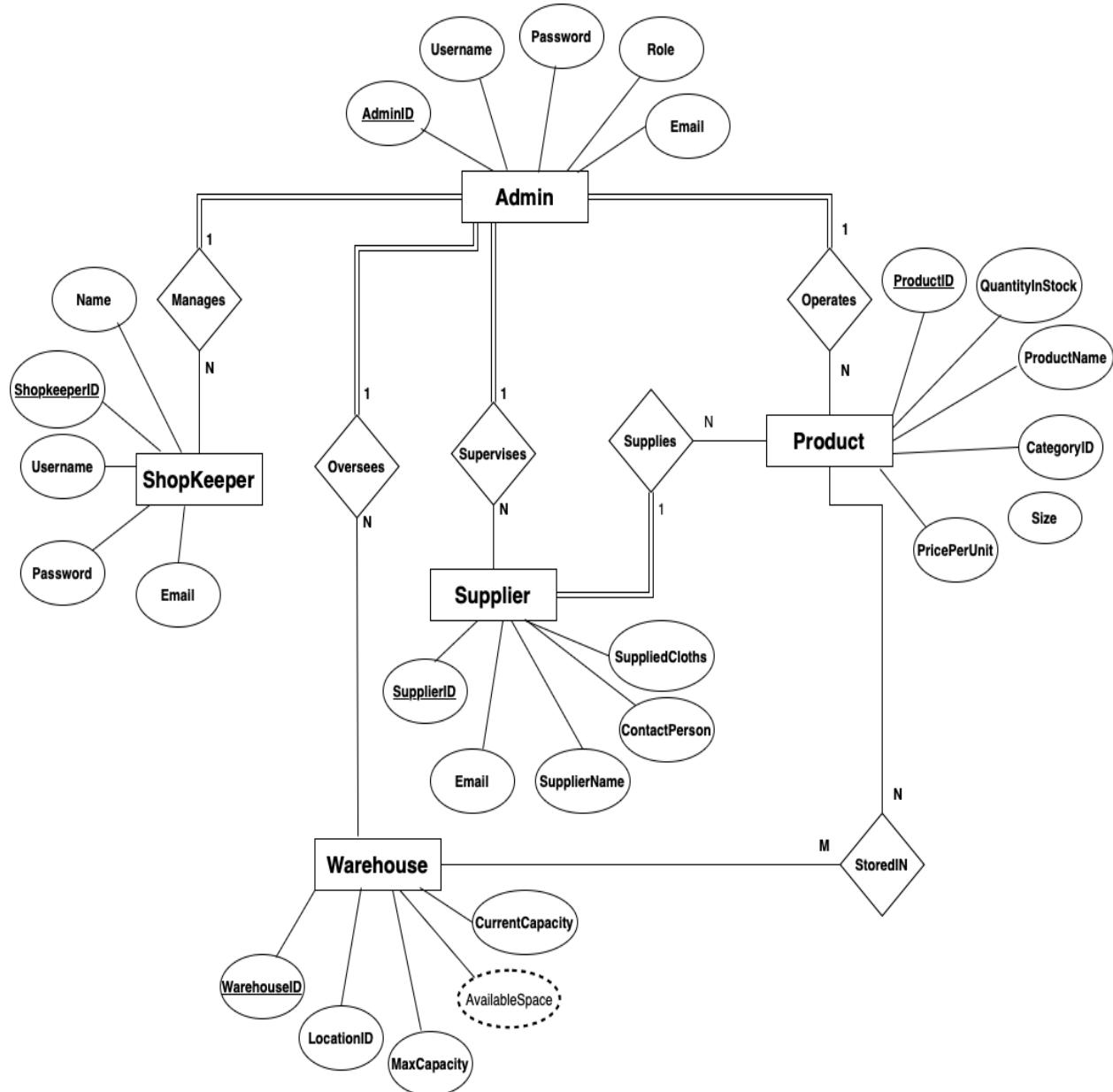


Figure 7. 3 External ER ADMIN POV [6]

The diagram places the Admin entity at the center, indicating a centralized management approach. The Admin has oversight of ShopKeepers, Suppliers, and

Products, implying broad control over the entire system. This centralized structure allows for efficient top-down management and decision-making processes.

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

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

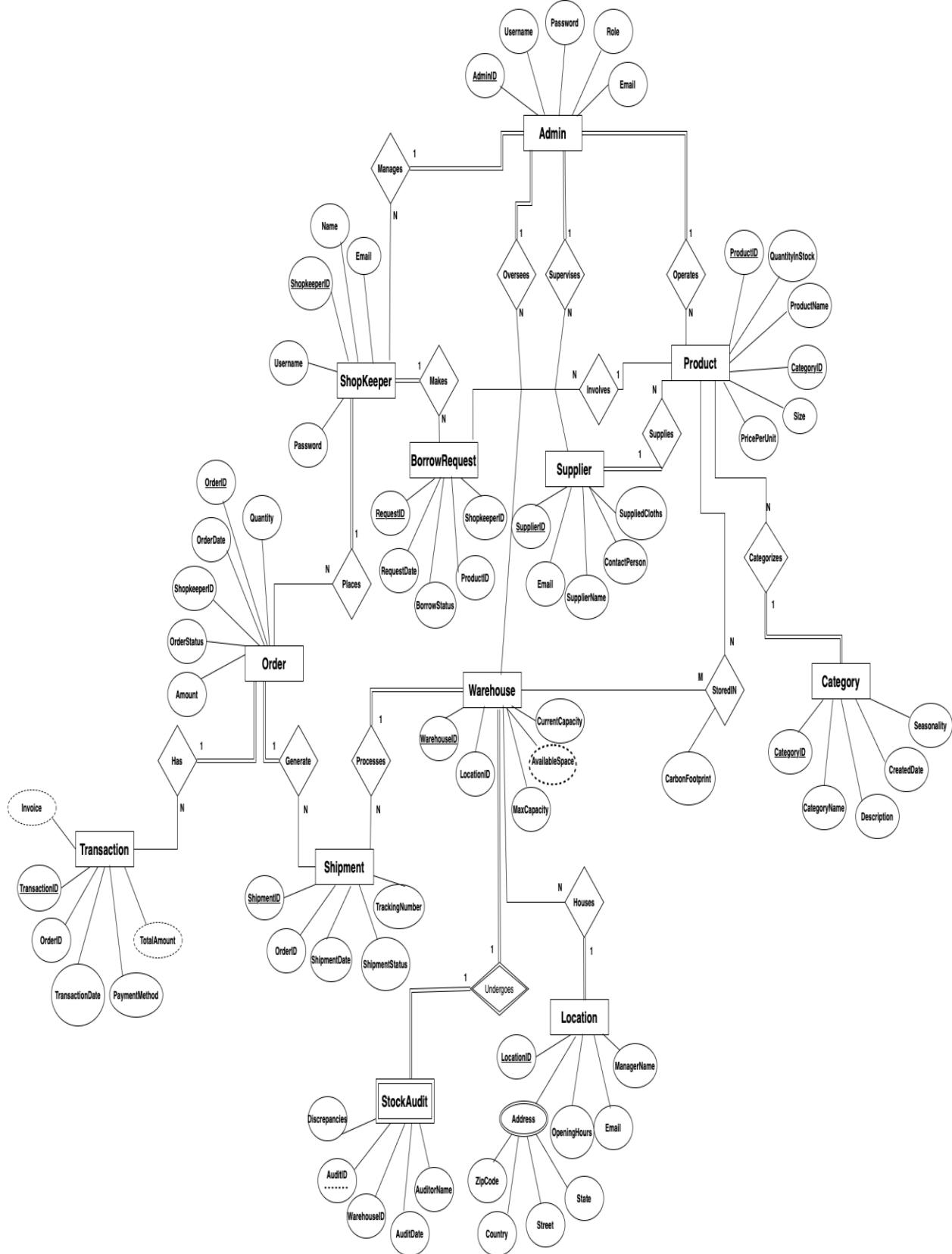


Figure 7. 4 Entity Relationship Diagram [6]

7.2 ENHANCED ENTITY RELATIONSHIP MODEL (EER MODEL)

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

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

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

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

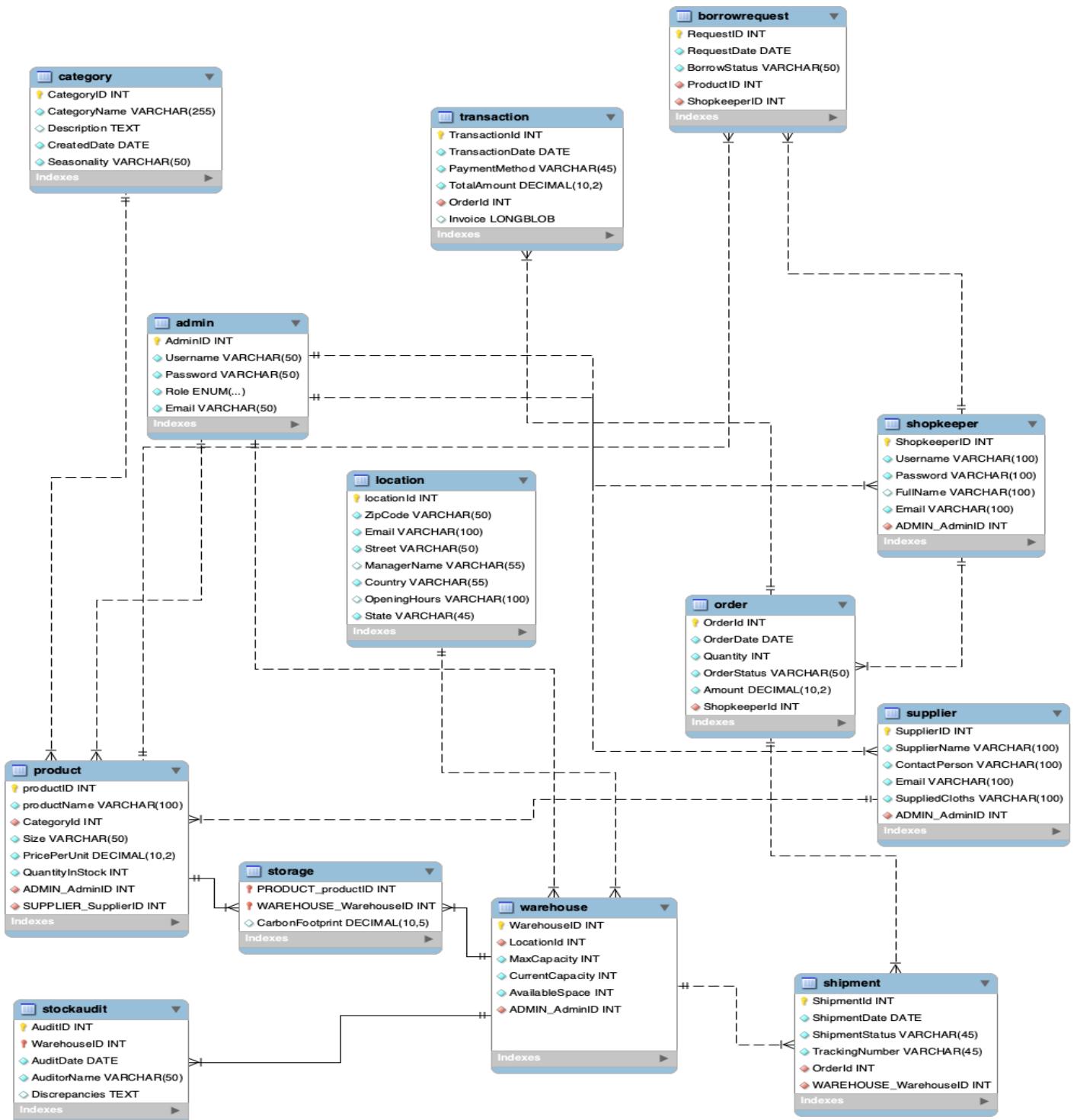


Figure 7.5 Enhanced Entity Relationship

Entities and Attributes [4][5]:

Entity: Admin

Description: administrator responsible for managing warehouses and overseeing operations.

Attributes:

AdminID (PK): Unique identifier for each admin

Username: Login name for the admin

Password: Encrypted password for account security

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

Email: Contact email for the admin

Entity: SHOPKEEPER

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

Attributes:

ShopkeeperID (PK): Unique identifier for each shopkeeper

Username: Login name for the shopkeeper

Password: Encrypted password for account security

FullName: Composite attribute (FirstName, LastName)

Email: Contact email for the shopkeeper

Entity: PRODUCT

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

Attributes:

productID (PK): Unique identifier for each product

productName: Name of the product

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

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

PricePerUnit: Cost of a single unit of the product

QuantityInStock: Current available quantity of the product

Entity: WAREHOUSE

Description: Represents storage facilities for products.

Attributes:

WarehouseID (PK): Unique identifier for each warehouse

LocationId (FK): References the location of the warehouse

MaxCapacity: Maximum storage capacity of the warehouse

CurrentCapacity: Current occupied capacity (derived)

AvailableSpace: Remaining storage space (derived)

Entity: BORROWREQUEST

Description: Represents requests from shopkeepers to borrow products.

Attributes:

RequestID (PK): Unique identifier for each request

RequestDate: Date when the request was made

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

ProductID (FK): References the product being requested

ShopkeeperID (FK): References the shopkeeper making the request

Entity: CATEGORY

Description: Represents different classifications of products.

Attributes:

CategoryID (PK): Unique identifier for each category

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

Description: Detailed description of the category

Createddate: Date when the category was created

Seasonality: Applicable season(s) for the category

Entity: SUPPLIER

Description: Represents companies or individuals who supply products.

Attributes:

SupplierID (PK): Unique identifier for each supplier

SupplierName: Name of the supplier company or individual

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

Email: Contact email for the supplier

SuppliedCloths: Multivalued attribute listing supplied product IDs

Entity: ORDER

Description: Represents customer purchases or stock orders.

Attributes:

orderId (PK): Unique identifier for each order

orderDate: Date when the order was placed

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

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

Amount: Total cost of the order

Entity: LOCATION

Description: Represents physical locations of warehouses or shops.

Attributes:

locationId (PK): Unique identifier for each location

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

PhoneNumber: Contact number for the location

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

OpeningHours: Multivalued attribute listing opening hours for each day

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

Entity: TRANSACTION

Description: Represents financial transactions related to orders.

Attributes:

TransactionID (PK): Unique identifier for each transaction

OrderID (FK): References the associated order

TransactionDate: Date when the transaction occurred

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

TotalAmount: Total amount of the transaction

Invoice: BLOB

Entity: STOCKAUDIT

Description: Represents periodic checks of warehouse inventory accuracy.

Attributes:

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

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

AuditDate: Date when the audit was conducted

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

Discrepancies: Multivalued attribute listing product IDs and quantity differences

Entity: SHIPMENT

Description: Represents the process of delivering orders to customers.

Attributes:

ShipmentID (PK): Unique identifier for each shipment

OrderID (FK): References the associated order

ShipmentDate: Date when the shipment was sent

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

TrackingNumber: Unique number for tracking the shipment

Multivalued Attributes:

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

Composite Attributes:

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

Derived Attributes:

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

Weak Entity:

- STOCKAUDIT (its existence depends on WAREHOUSE)

Strong Entities:

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

Total Participations:

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

Partial Participations:

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

Cardinality Ratios:

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

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

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

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

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

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

8. PRESENTATION SUBJECT

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

String Data Types:

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

4. **VARBINARY:** Variable length binary data.[10]
 - Similar to VARCHAR, except it stores binary strings instead of nonbinary strings.
 - Unlike CHAR and VARCHAR, the length of BINARY and VARBINARY is measured in 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:

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

```
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 accomodate 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 • TRUNCATE TABLE admin /username age location email profile_picture document bio status preferences unique_id
```

Output

Action Output

#	Time	Action	Message	Duration / Fetc
1	13:12:29	Drop database if exists Phase3	1 row(s) affected	0.015 sec
2	13:12:29	CREATE DATABASE Phase3	1 row(s) affected	0.000 sec
3	13:12:29	USE Phase3	0 row(s) affected	0.000 sec
4	13:12:29	CREATE TABLE admin (username VARCHAR(50), -- variable length string age INT, location VARCHAR(100), email VARCHAR(100), profile_picture BLOB, -- Binary Large Objects (e.g., images) document VARBINARY(65535), -- variable length binary data (e.g., files) bio TEXT, -- TEXT can accomodate large text data status ENUM('Active', 'Inactive', 'Pending'), -- predefined set of values preferences SET('Email', 'SMS', 'Push Notification'), -- SET is used for a collection of values unique_id BINARY(16) -- fixed length binary data);	0 row(s) affected	0.016 sec

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

```
21 • INSERT INTO admin (username, age, location, email, profile_picture, document, bio, status, preferences, unique_id)
22 VALUES (
23     'Sumanth',
24     24,
25     'New York',
26     'sumanth@gmail.com',
27     '0x89504E470D0A1A00000000D494844520000010000000100000001F4FB0D61', -- Example BLOB data
28     '0x255044620312E35000A25E2E3CF03000A312030206F626A00A3C3C2F4C656E67746B2034', -- Example VARBINARY data
29     'Intro of sumanth', -- Example TEXT data
30     'Active', -- Status ENUM
31     'Email,SMS', -- Preferences SET
32     UNHEX('4F6A1F8E90D6F4E88C9A8F2C504E6F70') -- Example BINARY data
33 );
34
35 • INSERT INTO admin (username, age, location, email, profile_picture, document, bio, status, preferences, unique_id)
36 VALUES (
37     'Abhijeet',
38     24,
39     'New York')
```

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_id)	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_id)	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_id)	1 row(s) affected	0.000 sec

String Functions:

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

64 • `SELECT username, LENGTH(username) AS username_length FROM admin; -- Length of string`

65

username	username_length
Sumanth	7
Abhijeet	8
Sagar	5

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid Form Editor Field Types

Result 2 x Read Only Context Help Snippets

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

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

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid Form Editor Field Types

Result 3 x Read Only Context Help Snippets

Action Output

#	Time	Action	Message	Duration / Fetch
1	13:18:11	SELECT username, LOWER(location) AS lower_location, UPPER(email) AS upper_email FROM admin LIMIT 0, ...	3 row(s) returned	0.000 sec / 0.000

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

Result 4 x

Output

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

Message

3 row(s) returned
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

Result 5 x

Output

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

Message

3 row(s) returned
3 row(s) returned
3 row(s) returned

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

MSCS_542L_256_24F: Project Progress Report: Phase05 TechBoys

```
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 | Filter Rows: Export: Wrap Cell Content: Result Grid
username short_location
Sumanth NY
Abhijeet NY
Sagar NY

Result 6 x Output 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

Result 7 x Output 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:

MSCS_542L_256_24F: Project Progress Report: Phase05 TechBoys

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

R2

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

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid | Form Editor | Field Types | Read Only | Context Help

admin 8 x

Output

Action Output

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

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

R4

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

Result Grid | Filter Rows: Export: Wrap Cell Content: Result Grid | Form Editor | Field Types | Read Only | Context Help

admin 9 x

Output

Action Output

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

8. Extracting values using SELECT:

MSCS_542L_256_24F: Project Progress Report: Phase05 TechBoys

```
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	sumanth@gmail.com			Intro of sumanth	Active	Email,SMS	
	Abhijeet	24	New York	abhi@gmail.com			Intro of Abhijeet	Pending	Email	

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: Phase05 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

```

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 insertion')	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

```

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 insertion')	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"}	Insertion using JSON_OBJECT()
["string2", 3, {"\key5": "value5", "\key6": "v..."}]	Insertion using JSON_ARRAY()
{"key1": ["string3", 6], "key2": "value2"}	Nested JSON: array in object

json_demo_table10 x

Output :

Action Output		
#	Time	Action
2	14:40:57	INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1": "value1", "key2": "value2"}', 'Direct...')
3	14:40:57	INSERT INTO json_demo_table(json_data, description) VALUES ([3, "string1"], 'Direct JSON array insertion')
4	14:41:43	INSERT INTO json_demo_table(json_data, description) VALUES (JSON_OBJECT("key3", "value3", "key4", ...))
5	14:41:44	INSERT INTO json_demo_table(json_data, description) VALUES (JSON_ARRAY("string2", 3, {"\key5": "value5", "\key6": "value6"}))
6	14:42:12	INSERT INTO json_demo_table(json_data, description) VALUES ('{"key1": ["string3", 6], "key2": "value2"}', 'Nested JSON: array in object')
7	14:42:23	SELECT * FROM json_demo_table LIMIT 0, 1000

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: Phase05 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"}')
[{"key1": "value1", "key2": "value2", "keymerge": "valuemerge"}]	[{"key1": "value1", "key2": "value2"}]	[{"key1": "value1", "key2": "value2"}]	[{"key1": "value1", "key2": "value2"}]	
[{"keymerge": "valuemerge"}, 3, "string1"]	[3, "string1"]	[3, "string1"]	[{"keymerge": "valuemerge"}]	
[{"key3": "value3", "key4": "value4", "keymerge": "valuemerge"}]	[{"key3": "value3", "key4": "value4"}]	[{"key3": "value3", "key4": "value4"}]	[{"key3": "value3", "key4": "value4"}]	
[{"key6": "value6"}]	[{"string2", 3, {"\key5": "value5", "\key6": "value6"}]}	[{"string2", 3, {"\key5": "value5", "\key6": "value6"}]}	[{"key1": [{"string3", 6}, "key2": "value2", "keymerge": "valuemerge"}]]	
"key2", "keymerge": "valuemerge", "key1": [{"string3", 6}, "key2": "value2", "keymerge": "valuemerge"}]]	[{"key1": [{"string3", 6}, "key2": "value2"}]]	[{"key1": [{"string3", 6}, "key2": "value2"}]]	[{"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"), 'JSON Object insertion')	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"}), 'JSON Array insertion')	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 insertion')	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),JSON_MERGE_PRESERVE(json_data, '{"keymerge": "valuemerge"}')	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"] {"key3": "value3", "key4": "value4"} ["string2", 3, {"\key5": "value5", "\key6": "value6"}] {"key1": [{"string3", 6}, "key2": "value2"]}	{"key1": "value1", "key2": "value2"} [3, ["string1", 1]] {"key3": "value3", "key4": "value4"} ["string2", [3, 1], {"\key5": "value5", "\key6": "value6"}] {"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

```

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with 'Result Grid', 'Filter Rows:', 'Export', and 'Wrap Cell Content'. Below the toolbar is a result grid table with one row. The table has two columns: 'json_data' and 'JSON_ARRAY_INSERT(json_data, '\$[0]', 1)'. The first column contains a complex JSON object, and the second column shows the result of inserting the value '1' at index 0. Below the result grid is a 'Result 13' section containing an 'Output' tab which displays the action history of the session.

json_data	JSON_ARRAY_INSERT(json_data, '\$[0]', 1)
{"key1": "value1", "key2": "value2"} [1, 3, "string1"] {"key3": "value3", "key4": "value4"} [1, "string2", 3, {"key5": "value5", "key6": "..."}] {"key1": ["string3", 6], "key2": "value2"}	{"key1": "value1", "key2": "value2"} [1, 3, "string1"] {"key3": "value3", "key4": "value4"} [1, "string2", 3, {"key5": "value5", "key6": "..."}] {"key1": [1, "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": "value5", "key6": "..."}), "Test JSON ARRAY")
6	14:42:12	INSERT INTO json_demo_table(json_data, description) VALUES ({"key1": ["string3", 6], "key2": "value2"}, "Test JSON ARRAY")
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)

```

The screenshot shows the MySQL Workbench interface. At the top, there's a toolbar with 'Result Grid', 'Filter Rows:', 'Export', and 'Wrap Cell Content'. Below the toolbar is a result grid table with two columns: 'json_data' and 'JSON_STORAGE_SIZE(json_data)'. The table lists five rows corresponding to the JSON objects in the 'json_demo_table'. The 'json_data' column shows the original JSON objects, and the 'JSON_STORAGE_SIZE(json_data)' column shows their respective storage sizes.

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

4. JSON_TYPE ():

```

59    -- Determining the type of JSON elements
60 •   SELECT json_data, JSON_TYPE(json_data)
61   FROM json_demo_table;
62
63    -- Getting the length of JSON elements (array length or number of key-value 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"');
69

```

```

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

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



Result 16 ×



Output:



Action 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: <input type="text"/> Export: Wrap Cell Content:	
json_data	JSON_REMOVE(json_data, '\$[0]')
{'key1': 'value1', 'key2': 'value2'}	{'key1': 'value1', 'key2': 'value2'}
[3, "string1"]	['string1']
{'key3': 'value3', 'key4': 'value4'}	{'key3': 'value3', 'key4': 'value4'}
['string2', 3, {"key5": "value5", "key6": "v..."}]	[3, {"key5": "value5", "key6": "value6"}]
{"key1": ["string3", 6], "key2": "value2"}	{"key1": ["string3", 6], "key2": "value2"}

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

```

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

```

Result Grid Filter Rows: <input type="text"/> Export: Wrap Cell Content:	
json_data	JSON_REPLACE(json_data, '\$.key1', 'val3')
{'key1': 'value1', 'key2': 'value2'}	{'key1': 'val3', 'key2': 'value2'}
[3, "string1"]	[3, "string1"]
{'key3': 'value3', 'key4': 'value4'}	{'key3': 'value3', 'key4': 'value4'}
['string2', 3, {"key5": "value5", "key6": "v..."}]	[3, {"key5": "value5", "key6": "value6"}]
{"key1": ["string3", 6], "key2": "value2"}	{"key1": ["string3", 6], "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_data	
	JSON_SET(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"] {"key1": "val3", "key3": "value3", "key4": "valu... ["string2", 3, {"key5": "value5", "key6": "v... {"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, 4. AdminID references the admin table. Foreign Key

	<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> <ol 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> <ol style="list-style-type: none"> 1. StockAuditID must be unique and cannot be null. 2. WarehouseID references the Warehouse table.

	<p>Relationships:</p> <ol 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>

	<p>1. CategoryID must be unique and cannot be null. PRIMARY KEY.</p> <p>Relationships:</p> <ul 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> <ul 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> <ul 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> <ul 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> <ol style="list-style-type: none"> 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 `fk_WAREHOUSE_ADMIN1`
        FOREIGN KEY (`LocationId`)
        REFERENCES
            `Warehouse_Management_System_DBMS_Project`.`LOCATION`(`locationId`),
    CONSTRAINT `fk_WAREHOUSE_ADMIN1`
        FOREIGN KEY (`ADMIN_AdminID`)
        REFERENCES
            `Warehouse_Management_System_DBMS_Project`.`ADMIN`(`AdminID`));
```

9. CREATE BORROWREQUEST TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`BORROWREQUEST` (
    `RequestID` INT NOT NULL AUTO_INCREMENT,
    `RequestDate` DATE NOT NULL,
    `BorrowStatus` VARCHAR(50) NOT NULL,
```

```
`ProductID` INT NOT NULL,  
`ShopkeeperID` INT NOT NULL,  
PRIMARY KEY (`RequestID`),  
CONSTRAINT ``  
    FOREIGN KEY (`ProductID`)  
        REFERENCES  
`Warehouse_Management_System_DBMS_Project`.`PRODUCT`  
(`productID`),  
CONSTRAINT ``  
    FOREIGN KEY (`ShopkeeperID`)  
        REFERENCES  
`Warehouse_Management_System_DBMS_Project`.`SHOPKEEPER`  
(`ShopkeeperID`);
```

10. CREATE STOCKAUDIT TABLE:

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

11. CREATE ORDER TABLE:

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

12. CREATE SHIPMENT TABLE:

```
CREATE TABLE IF NOT EXISTS
`Warehouse_Management_System_DBMS_Project`.`SHIPMENT` (
    `ShipmentId` INT NOT NULL AUTO_INCREMENT,
    `ShipmentDate` DATE NOT NULL,
    `ShipmentStatus` VARCHAR(45) NOT NULL,
    `TrackingNumber` VARCHAR(45) NOT NULL,
    `OrderId` INT NOT NULL,
    `WAREHOUSE_WarehouseID` INT NOT NULL,
    PRIMARY KEY (`ShipmentId`),
    CONSTRAINT `OrderId`
        FOREIGN KEY (`OrderId`)
```

REFERENCES

```
`Warehouse_Management_System_DBMS_Project`.`ORDER`(`OrderId`),
CONSTRAINT `fk_Shipment_WAREHOUSE1`
FOREIGN KEY (`WAREHOUSE_WarehouseID`)
```

REFERENCES

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

13. CREATE STORAGE TABLE:

CREATE TABLE IF NOT EXISTS

```
`Warehouse_Management_System_DBMS_Project`.`STORAGE` (
`PRODUCT_productID` INT NOT NULL AUTO_INCREMENT,
`WAREHOUSE_WarehouseID` INT NOT NULL,
`CarbonFootprint` DECIMAL(10,5) NULL,
PRIMARY KEY (`PRODUCT_productID`, `WAREHOUSE_WarehouseID`),
CONSTRAINT `fk_PRODUCT_has_WAREHOUSE_PRODUCT1`
FOREIGN KEY (`PRODUCT_productID`)
```

REFERENCES

```
`Warehouse_Management_System_DBMS_Project`.`PRODUCT`(`productID`),
CONSTRAINT `fk_PRODUCT_has_WAREHOUSE_WAREHOUSE1`
FOREIGN KEY (`WAREHOUSE_WarehouseID`)
```

REFERENCES

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

14. CREATE TRANSACTION TABLE:

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

10.LOADING DATA AND PERFORMANCE ENHANCEMENT

10.1 HANDLING KEY CONSTRAINTS:

Case 1: Foreign Key Constraint

- a) Insert parent ADMIN records first

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

- b) Then insert child SHOPKEEPER records

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

- c) Insert second value in shopkeeper

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

- d) Error Occurred

```
Error Code: 1452. Cannot add or update a child row: a
foreign key constraint fails
(`warehouse_management_system_dbms_project`.`shopkeeper`
```

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

e) Disable FK checks

```
SET foreign_key_checks = 0;
```

f) Insert SHOPKEEPER records with values without foreign key issue

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

g) Enable FK checks again

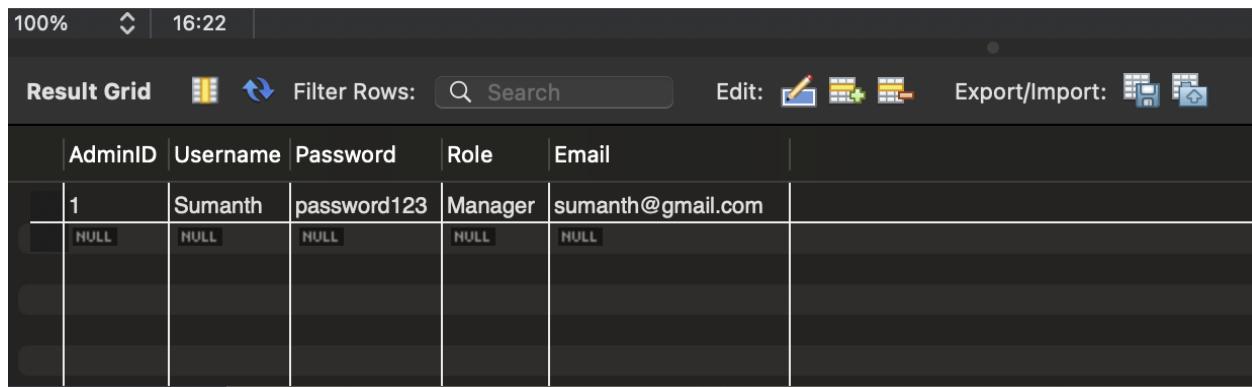
```
SET foreign_key_checks = 1;
```

Description:

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

Case 2: Primary Key

- a) Current Shopkeeper records first



AdminID	Username	Password	Role	Email	
1	Sumanth	password123	Manager	sumanth@gmail.com	
NULL	NULL	NULL	NULL	NULL	
NULL	NULL	NULL	NULL	NULL	
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  
FIELDS TERMINATED BY ','  
LINES TERMINATED BY '\n';
```

MSCS_542L_256_24F: Project Progress Report: Phase05 TechBoys

6	michael_d davismich	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_ga garciajessi	Manager	jessica.garcia@gmail.com
10	kevin_rodr rodriguezk	Supervisor	kevin.rodriguez@yahoo.com
11	ashley_ma martinezai	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 jacksonanj	Manager	angela.jackson@gmail.com
16	daniel_har harrisdanic	Supervisor	daniel.harris@yahoo.com
17	amanda_lk lewisaman	Manager	amanda.lewis@gmail.com
18	joshua_sc scottjoshu	Supervisor	joshua.scott@yahoo.com
19	nicole_gre greennicol	Manager	nicole.green@gmail.com
20	justin_bak bakerjustir	Supervisor	justin.baker@yahoo.com
21	rebecca_a 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_mitc 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';

6	Jessica	shopper30	Jessica Ga	jessica.gar	6
7	Kevin	shopper40	Kevin Rodr	kevin.rodr	7
8	Ashley	shopper50	Ashley Ma	ashley.ma	8
9	Christophe	shopper60	Christophe	christophe	9
10	Sarah	shopper70	Sarah Tho	sarah.tho	10
11	Matthew	shopper80	Matthew \	matthew.v	11
12	Angela	shopper90	Angela Jac	angela.jac	12
13	Daniel	shopper10	Daniel Har	daniel.har	13
14	Amanda	shopper12	Amanda Le	amanda.le	14
15	Joshua	shopper13	Joshua Sc	joshua.scc	15
16	Nicole	shopper14	Nicole Gre	nicole.gre	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.mitc	21
22	Stephanie	shopper20	Stephanie	stephanie.	22
23	Jason	shopper21	Jason Rob	jason.robe	23
24	Katherine	shopper22	Katherine	katherine.	24
25	Timothy	shopper23	Timothy Pl	timothy.ph	25
26	Elizabeth	shopper24	Elizabeth E	elizabeth.e	26
27	Ryan	shopper25	Ryan Colli	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.doe@gmail.com')	1 row(s) affected	0.031 sec

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

```

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', 'lealee456', '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 screenshot shows the MySQL Workbench interface. On the left, the SQL editor contains the provided bulk insert script. On the right, the 'Output' tab displays the execution log. The log shows three actions: setting FOREIGN_KEY_CHECKS to 0 (0 rows affected), truncating the ADMIN table (0 rows affected), and inserting 5 rows into the ADMIN table (5 rows affected). The total duration for the entire operation was 0.016 seconds.

#	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.d... 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', 'leelalice456', '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'),
```

MSCS_542L_256_24F: Project Progress Report: Phase05 TechBoys

```
('Spring Collection', 'Floral and light spring designs.', '2023-03-01', 'Spring'),  
('Fall Collection', 'Warm autumn colors and designs.', '2023-09-01', 'Fall'),  
(('Accessories', 'Various fashion accessories for all seasons.', '2023-05-15', 'All Seasons'));
```

SUPPLIER insert statements:

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

LOCATION insert statements:

```
INSERT INTO LOCATION (ZipCode, Email, Street, ManagerName, Country, OpeningHours, State) VALUES  
(('10001', 'john@warehouse.com', '5th Ave', 'John Snow',  
'USA', '9 AM - 5 PM', 'NY'),  
(('20002', 'lannister@warehouse.com', 'Broadway St',  
'Cersei Lannister', 'USA', '10 AM - 6 PM', 'CA'),
```

```
('30003', 'targaryen@warehouse.com', 'Elm St',
'Daenerys Targaryen', 'USA', '8 AM - 4 PM', 'TX'),
('40004', 'stark@warehouse.com', 'Main St', 'Arya
Stark', 'USA', '7 AM - 3 PM', 'FL'),
('50005', 'jeff@warehouse.com', 'Sunset Blvd', 'Jeff
Bezos', 'USA', '11 AM - 7 PM', 'NV');
```

PRODUCT insert statements:

```
INSERT INTO PRODUCT (productName, CategoryId, Size,
PricePerUnit, QuantityInStock, ADMIN_AdminID,
SUPPLIER_SupplierID) VALUES
('Cotton Shirt', 1, 'L', 19.99, 50, 1, 1),
('Woolen Jacket', 2, 'M', 49.99, 30, 2, 2),
('Silk Scarf', 5, 'One Size', 14.99, 100, 3, 3),
('Denim Jeans', 1, 'XL', 39.99, 40, 4, 4),
('Leather Gloves', 2, 'M', 24.99, 20, 5, 5);
```

WAREHOUSE insert statements:

```
INSERT INTO WAREHOUSE (LocationId, MaxCapacity,
CurrentCapacity, AvailableSpace, ADMIN_AdminID) VALUES
(1, 1000, 800, 200, 1),
(2, 1500, 1000, 500, 2),
(3, 2000, 1800, 200, 3),
(4, 1200, 900, 300, 4),
(5, 1700, 1500, 200, 5);
```

BORROWREQUEST insert statements:

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

STOCKAUDIT insert statements:

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

ORDER insert statements:

```
INSERT INTO  
Warehouse_Management_System_DBMS_Project.order  
(OrderDate, Quantity, OrderStatus, Amount,  
ShopkeeperId) VALUES  
('2024-09-01', 10, 'Shipped', 199.90, 1),
```

```
('2024-09-02', 5, 'Delivered', 99.95, 2),  
('2024-09-03', 20, 'Processing', 399.80, 3),  
('2024-09-04', 15, 'Canceled', 299.85, 4),  
('2024-09-05', 8, 'Pending', 159.92, 5);
```

SHIPMENT insert statements:

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

TRANSACTION insert statements:

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

STORAGE insert statements:

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

10.4 INDEX OPTIMIZATION

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

INDEX ON ADMIN:

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

INDEX ON SHOPKEEPER:

```
CREATE INDEX idx_shopkeeper_username ON  
SHOPKEEPER(Username);  
CREATE INDEX idx_shopkeeper_email ON SHOPKEEPER>Email);
```

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

INDEX ON CATEGORY:

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

INDEX ON SUPPLIER:

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

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

INDEX ON LOCATION:

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

```
CREATE INDEX idx_location_country ON LOCATION(Country);
```

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

INDEX ON PRODUCT:

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

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

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

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

INDEX ON WAREHOUSE:

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

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

INDEX ON BORROWREQUEST:

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

```
CREATE INDEX idx_borrowrequest_product ON  
BORROWREQUEST(ProductID);
```

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

INDEX ON ORDER:

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

```
CREATE INDEX idx_order_status ON `ORDER` (OrderStatus);
```

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

INDEX ON SHIPMENT:

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

```
CREATE INDEX idx_shipment_status ON  
SHIPMENT(ShipmentStatus);
```

```
CREATE INDEX idx_shipment_order ON SHIPMENT(OrderId);
```

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

INDEX ON TRANSACTION:

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

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

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

Query:

```
SELECT s.ShopkeeperID, s.Username, p.productName,  
c.CategoryName  
FROM SHOPKEEPER s JOIN  
PRODUCT p ON s.ShopkeeperID = p.ADMIN_AdminID  
JOIN  
CATEGORY c ON p.CategoryId = c.CategoryID  
WHERE  
s.Username = 'DAVID';
```

EXPLAIN Plan before INDEX Creation.

MSCS_542L_256_24F: Project Progress Report: Phase05 TechBoys

```
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)
```

11. REFERENCES

[1] Zoho Inventory

[2] Fishbowl Inventory

[3] Korber Warehouse Management Systems

[4] Warehouse management system - ER Diagram

[5] Design ER Diagrams for Supply Chain Management

[6] Draw.io

[7] SQL data types

[8] String type syntax

[9] Char

[10] Binary and Varbinary

[11] Blob

[12] Enum

[13] Set

[14] JSON data types

[15] Column-indexes

[16] Optimization-indexes

[17] Insert-optimization