Optimizing Transportation Logistics with TMS

**Advanced Database Design for Fleet Management and Performance Analytics**

A diagram of a truck

Description automatically generated

**Marckenrold Cadet**

**Zhi Zheng**

**David Carbajal**

06.18.2024

Florida Polytechnic University

Advanced Database System Design

Instructor: Karim Elish

# Introduction: Overview of the System Requirements

The project focuses on building a Transportation Management System (TMS) to optimize transportation logistics for businesses. The TMS will include functionalities such as route optimization, fleet management, scheduling, tracking, and reporting. The database management system chosen for this project is Oracle.

Project Objectives:

The project's objective outlined in the provided source is to allow students to apply database design concepts to solve transportation logistics problems using a Transportation Management System (TMS). The TMS will be a comprehensive software solution that optimizes transportation logistics. Students will design and implement a relational database to store information such as customers, orders, vehicles, drivers, routes, schedules, and historical data using a database management system. They will also develop a user-friendly web-based interface with features like user authentication, role-based access, and interactive dashboards. Finally, the project requires implementing reporting functionalities for analytics reports (e.g., delivery performance, fuel consumption, and customer satisfaction) and visualizations.

Project Scope:

The project focuses on developing a Transportation Management System (TMS), a software solution designed to optimize transportation logistics for businesses. The TMS will incorporate functionalities such as:

1. Route optimization
2. Scheduling
3. Tracking
4. Reporting

The project involves designing and implementing a robust relational database to manage information related to customers, orders, vehicles, drivers, routes, schedules, and historical data. This will require creating a user-friendly web-based interface for administrators, dispatchers, drivers, and customers, including features for user authentication, role-based access control, and interactive dashboards. The project also includes implementing reporting functionalities to generate analytics reports, such as delivery performance metrics, fuel consumption analysis, cost analysis, and customer satisfaction ratings. These reports will be presented using visualizations such as charts and graphs to highlight key performance indicators and trends.

# Detailed Database Business Rules & Assumptions

*Business Rules:*

The project overview provided outlines the need for a robust relational database to support a Transportation Management System (TMS). This database will store information about various entities such as customers, orders, vehicles, drivers, routes, schedules, and historical data. The following business rules are defined to ensure the integrity and efficiency of the TMS:

1. Customer Management
   1. Unique customer ID.
   2. Detailed customer information.
   3. Multiple orders per customer.
2. Order Management
   1. Unique order ID.
   2. Orders linked to customer IDs.
   3. Order details and delivery routes.
3. Vehicle Management
   1. Unique vehicle ID.
   2. Detailed vehicle information.
   3. Vehicle assignment to drivers and routes.
4. Driver Management
   1. Unique driver ID.
   2. Detailed driver information.
   3. Valid licenses and certifications.
5. Route Management
   1. Unique route ID.
   2. Route details and optimization.
   3. Association with multiple orders and vehicles.
6. Schedule Management
   1. Unique schedule ID.
   2. Schedule details considering availability
7. Historical Data Management
   1. Past orders, routes, vehicle usage, and driver performance.
8. User Authentication and Access Control
   1. Unique user IDs and secure passwords.
   2. Role-based access control.
   3. Activity logs.
9. Reporting and Analytics
   1. Generate and customize reports.
   2. Data visualization tools.
10. Compliance and Data Security
    1. Compliance with industry standards and regulations.
    2. Data encryption.
    3. Regular security audits.

*Assumptions:*

The design and implementation of the Transportation Management System (TMS) were based on several key assumptions. User and role management assumed that each user in the system has a unique UserID and is assigned specific roles (Customer, Driver, Administrator, Dispatcher), which determine their access and permissions within the TMS. User credentials, including Username and Password, are securely stored and managed.

Data integrity was ensured by defining primary keys for all tables to uniquely identify each record and enforcing foreign key constraints to maintain referential integrity between related tables, such as linking orders to customers and shipments to orders. The database schema was designed to be in at least Third Normal Form (3NF) to eliminate redundancy and ensure logical data dependencies.

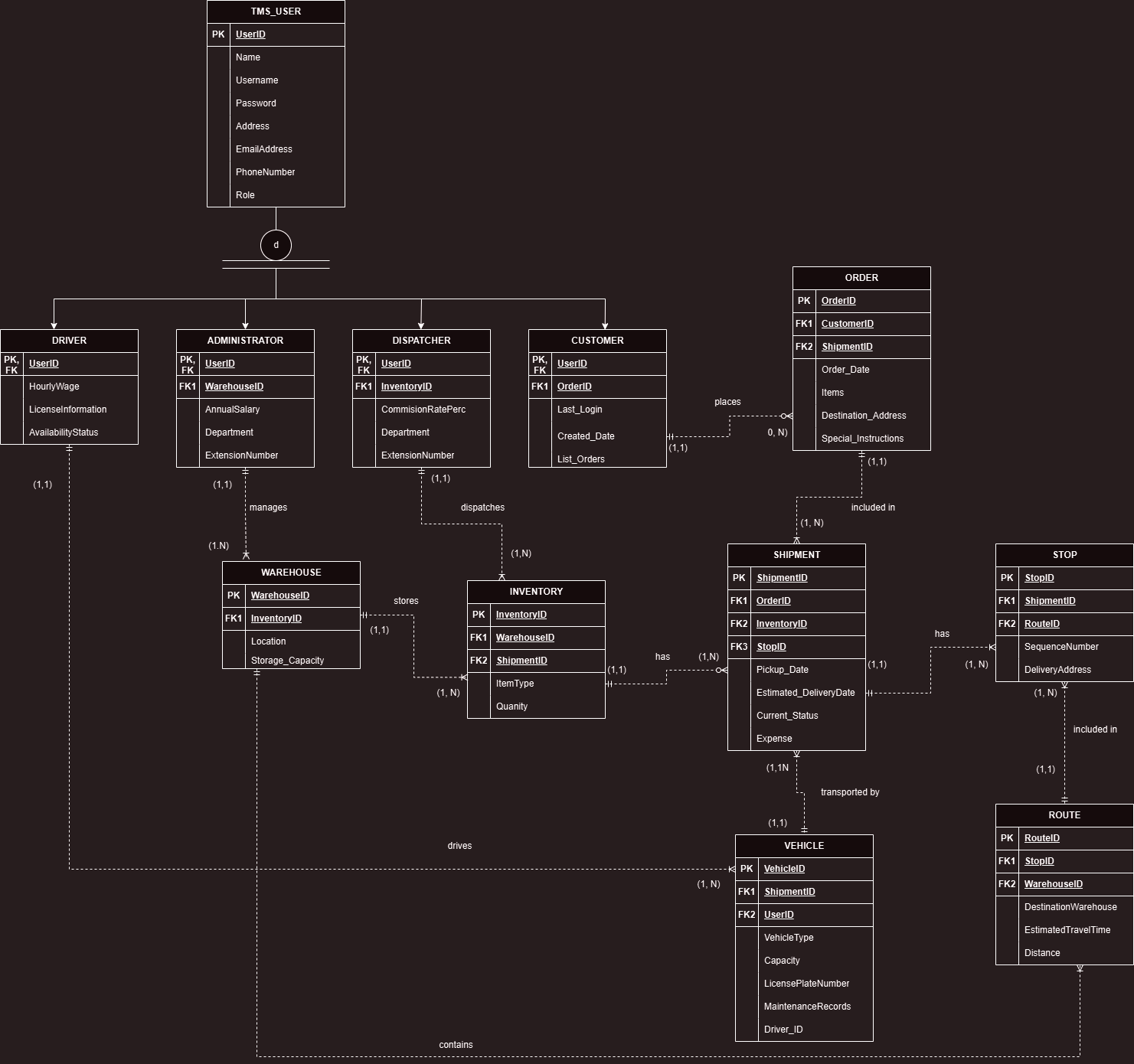
Warehouses (WAREHOUSE), inventory (INVENTORY), administrators (ADMINISTRATOR), drivers (DRIVER), dispatchers (DISPATCHER), orders (ORDER), shipments (SHIPMENT), stops (STOP), routes (ROUTE), and vehicles (VEHICLE), with appropriate data types and constraints to ensure data validity. Initial data insertion provided a basis for testing and demonstration, including sample records for users, customers, warehouses, inventory items, administrators, drivers, dispatchers, orders, shipments, stops, routes, and vehicles.

Indexing was applied to frequently queried columns to optimize data retrieval performance, including indexes on columns like Name, Username, EmailAddress in TMS\_USERS, Last\_Login and Created\_Date in CUSTOMER, Location in WAREHOUSE, and other relevant columns in various tables. Foreign key relationships were established to link related tables, ensuring consistent data references across the database.

It was assumed that data entered into the system would be accurate and consistent, with regular data validation and integrity checks performed to maintain data quality. The system supports real-time operations, such as tracking and monitoring shipments, updating driver availability status, and adjusting warehouse inventory levels.

Security and compliance measures included implementing user authentication to ensure secure access to the system, designing the system to comply with relevant data protection and privacy regulations, and securely storing sensitive data, such as passwords, using encryption. The database schema was also designed to accommodate future growth in the number of users, orders, and vehicles, ensuring the system can scale horizontally and vertically to handle increased load and data volume.

# Entity Relationship (ER) Model: Detailed Description

Diagram 1: TMS ERD

#### Entities and Attributes

* 1. *TMS\_USER:* encompasses a range of attributes essential for user management within the system. These attributes include UserID, which serves as the Primary Key (PK) ensuring unique identification of each user. Other attributes are Name, Username, Password, Address, EmailAddress, and PhoneNumber, capturing essential personal and contact information. Additionally, the Role attribute functions as a discriminator to differentiate between various subtypes of users such as DRIVER, ADMINISTRATOR, DISPATCHER, and CUSTOMER, enabling the system to manage and apply specific attributes and functionalities pertinent to each user type.
  2. *DRIVER:* includes a range of attributes specific to managing driver information within the system. These attributes include UserID, which serves as both the Primary Key (PK) and a Foreign Key (FK) referencing the TMS\_USER entity, ensuring unique identification and relational integrity. Additional attributes are HourlyWage, LicenseInformation, and AvailabilityStatus, capturing essential details about the driver's compensation, licensure, and availability for assignments.
  3. *ADMINISTRATOR:* encompasses attributes critical for managing administrative roles within the system. These attributes include UserID, serving as both the Primary Key (PK) and a Foreign Key (FK) referencing TMS\_USER, ensuring unique identification and relational integrity. Other attributes are WarehouseID, which is a Foreign Key (FK) referencing the WAREHOUSE entity, AnnualSalary, Department, and ExtensionNumber, capturing essential details about the administrator's assigned warehouse, compensation, department, and contact information.
  4. *DISPATCHER:* includes attributes essential for managing dispatch operations within the system. These attributes include UserID, serving as both the Primary Key (PK) and a Foreign Key (FK) referencing TMS\_USER, ensuring unique identification and relational integrity. Additional attributes are InventoryID, which is a Foreign Key (FK) referencing the INVENTORY entity, CommissionRatePerc, Department, and ExtensionNumber, capturing crucial details about the dispatcher's commission rate, department affiliation, and contact extension.
  5. *CUSTOMER:* encompasses attributes necessary for managing customer information within the system. These attributes include UserID, which serves as both the Primary Key (PK) and a Foreign Key (FK) referencing TMS\_USER, ensuring unique identification and relational integrity. Other attributes are Last\_Login, Created\_Date, and List\_Orders, capturing essential details about the customer's last login time, account creation date, and the list of orders placed.
  6. *WAREHOUSE:* includes attributes critical for managing warehouse operations within the system. These attributes include WarehouseID, which serves as the Primary Key (PK) ensuring unique identification of each warehouse. Additional attributes are Location and Storage\_Capacity, capturing essential details about the warehouse's geographical location and its storage capacity.
  7. *INVENTORY:* encompasses attributes essential for managing inventory items within the system. These attributes include InventoryID, which serves as the Primary Key (PK) ensuring unique identification of each inventory item. Other attributes are WarehouseID, a Foreign Key (FK) referencing the WAREHOUSE entity, ItemType, and Quantity, capturing essential details about the type of inventory item and its quantity in stock.
  8. *ORDER:* includes attributes critical for managing customer orders within the system. These attributes include OrderID, which serves as the Primary Key (PK) ensuring unique identification of each order. Additional attributes are CustomerID, a Foreign Key (FK) referencing the CUSTOMER entity, Order\_Date, Items, Destination\_Address, and Special\_Instructions, capturing essential details about the order date, items included, destination address, and any special instructions.
  9. *SHIPMENT:* encompasses attributes essential for managing shipments within the system. These attributes include ShipmentID, which serves as the Primary Key (PK) ensuring unique identification of each shipment. Other attributes are OrderID, a Foreign Key (FK) referencing the ORDER entity, Pickup\_Date, Estimated\_DeliveryDate, Expense, and Current\_Status, capturing crucial details about the pickup date, estimated delivery date, shipment expense, and its status.
  10. *STOP:* includes attributes necessary for managing stops within a shipment route in the system. These attributes include StopID, which serves as the Primary Key (PK) ensuring unique identification of each stop. Additional attributes are ShipmentID, a Foreign Key (FK) referencing the SHIPMENT entity, RouteID, a Foreign Key (FK) referencing the ROUTE entity, SequenceNumber, and DeliveryAddress, capturing essential details about the stop's sequence in the route and its delivery address.
  11. *ROUTE:* encompasses attributes critical for managing transportation routes within the system. These attributes include RouteID, which serves as the Primary Key (PK) ensuring unique identification of each route. Other attributes are WarehouseID, a Foreign Key (FK) referencing the WAREHOUSE entity, DestinationWarehouse, EstimatedTravelTime, and Distance, capturing essential details about the destination warehouse, estimated travel time, and route distance.
  12. *VEHICLE:* includes attributes essential for managing vehicles within the system. These attributes include VehicleID, which serves as the Primary Key (PK) ensuring unique identification of each vehicle. Additional attributes are Capacity, LicensePlateNumber, MaintenanceRecords, and DriverID, a Foreign Key (FK) referencing the DRIVER entity, capturing crucial details about the vehicle's capacity, license plate number, maintenance records, and associated driver.

#### Keys

Primary Keys (PK): Unique identifiers for each entity.

1. TMS\_USER: UserID
2. DRIVER: UserID
3. ADMINISTRATOR: UserID
4. DISPATCHER: UserID
5. CUSTOMER: UserID
6. WAREHOUSE: WarehouseID
7. INVENTORY: InventoryID
8. ORDER: OrderID
9. SHIPMENT: ShipmentID
10. STOP: StopID
11. ROUTE: RouteID
12. VEHICLE: VehicleID

Foreign Keys (FK): Ensure relational integrity.

1. DRIVER: UserID (references TMS\_USER)
2. ADMINISTRATOR: UserID (references TMS\_USER), WarehouseID (references WAREHOUSE)
3. DISPATCHER: UserID (references TMS\_USER), InventoryID (references INVENTORY)
4. CUSTOMER: UserID (references TMS\_USER)
5. INVENTORY: WarehouseID (references WAREHOUSE)
6. ORDER: CustomerID (references CUSTOMER)
7. SHIPMENT: OrderID (references ORDER)
8. STOP: ShipmentID (references SHIPMENT), RouteID (references ROUTE)
9. ROUTE: WarehouseID (references WAREHOUSE)
10. VEHICLE: DriverID (references DRIVER)

A comprehensive data dictionary, detailing all entities, attributes, keys, and relationships within the system, can be found in the appendix. This data dictionary includes a detailed table for each entity with columns specifying the attribute name, data type, constraints, and a description of each attribute. This structured approach ensures clarity and consistency in understanding the database schema, providing a valuable reference for all stakeholders involved in data management and system operations.

#### Relationships, Cardinality, and Connectivity

* TMS\_USER and DRIVER, ADMINISTRATOR, DISPATCHER, CUSTOMER: The relationship between TMS\_USER and its subtypes (DRIVER, ADMINISTRATOR, DISPATCHER, CUSTOMER) is a one-to-one, strong relationship with TMS\_USER, acting as a supertype for these roles. Each subtype has a foreign key relationship back to TMS\_USER, ensuring that each specific user role inherits attributes from TMS\_USER while adding its unique attributes. This setup uses a disjoint and total specialization constraint, where each user must belong to only one specific subtype, and every user must belong to one of these subtypes.
* *ADMINISTRATOR and WAREHOUSE:* One-to-many, weak relationship, with each administrator managing one or more warehouses. This ensures that each warehouse has a corresponding administrator responsible for its operations.
* *WAREHOUSE and INVENTORY:* One-to-many, weak relationship, with each warehouse storing multiple inventory items. This structure supports efficient management and tracking of inventory across different warehouses.
* *INVENTORY and SHIPMENT:* Many-to-one, weak relationship, with each inventory item being included in one shipment, but a shipment containing multiple inventory items. This relationship ensures that inventory items are properly grouped and managed within shipments.
* *CUSTOMER and ORDER:* One-to-many, weak relationship, with each customer placing multiple orders. This reflects typical customer behavior in a transactional system, ensuring that all orders placed by a customer are properly recorded and managed.
* *ORDER and SHIPMENT:* One-to-many, weak relationship, with each order being included in multiple shipments. This allows for flexible shipping arrangements and partial order fulfillments if necessary.
* *SHIPMENT and STOP:* One-to-many, weak relationship, with each shipment including multiple stops. This structure supports detailed tracking of shipment progress and delivery points.
* *STOP and ROUTE:* One-to-many, weak relationship, with each stop being part of one route, but a route having multiple stops. This relationship ensures efficient route planning and tracking.
* *SHIPMENT and VEHICLE:* One-to-many, weak relationship, with each shipment being transported by multiple vehicles. This allows for the distribution of shipment loads across several vehicles to optimize transportation efficiency.
* *DRIVER and VEHICLE:* One-to-many, weak relationship, with each driver being associated with multiple vehicles. This supports scenarios where drivers may operate different vehicles over time or manage a fleet of vehicles.
* *DISPATCHER and INVENTORY:* One-to-many, weak relationship, with each dispatcher managing multiple inventory items. This ensures that inventory management tasks are properly assigned to dispatchers.
* *SHIPMENT and ORDER:* One-to-one, weak relationship, with each shipment being linked to one specific order. This relationship ensures accurate tracking of orders within shipments.
* *ROUTE and WAREHOUSE:* One-to-many, weak relationship, with each route originating from or connecting to multiple warehouses. This structure supports efficient logistical planning and execution.

# Normalization: How the Design Satisfies 3NF

The database schema for the Transportation Management System (TMS) was meticulously designed to adhere to the principles of normalization, ensuring that all tables are in at least Third Normal Form (3NF). This design strategy maintains data integrity and reduces redundancy by ensuring that each table has a primary key, and all non-key columns depend directly on the primary key without any transitive dependencies or partial dependencies.

The TMS\_USER table uses UserID as the primary key. All columns in this table, such as Name, Username, Password, Address, EmailAddress, PhoneNumber, and Role, depend directly on this key without any transitive dependencies or partial dependencies. Similarly, the CUSTOMER, DRIVER, ADMINISTRATOR, and DISPATCHER tables also use UserID as their primary key, which is a foreign key referencing the TMS\_USER table. In these tables, all columns depend directly on UserID, and there are no transitive dependencies or partial dependencies. Additionally, the ADMINISTRATOR table includes a foreign key WarehouseID, and the DISPATCHER table includes a foreign key InventoryID.

The WAREHOUSE table uses WarehouseID as the primary key, with all columns, such as Location and Storage\_Capacity, depending directly on this key and having no transitive dependencies or partial dependencies. The INVENTORY table uses InventoryID as the primary key and includes a foreign key WarehouseID. All columns in the INVENTORY table depend directly on InventoryID without any transitive dependencies or partial dependencies.

The ORDER table uses OrderID as the primary key and includes a foreign key CustomerID. All columns, such as Order\_Date, Items, Destination\_Address, and Special\_Instructions, depend directly on OrderID, with no transitive dependencies or partial dependencies. The SHIPMENT table uses ShipmentID as the primary key and includes foreign keys OrderID and InventoryID. All columns in the SHIPMENT table depend directly on ShipmentID, and there are no transitive dependencies or partial dependencies.

The STOP table uses StopID as the primary key and includes foreign keys ShipmentID and RouteID. All columns, such as SequenceNumber and DeliveryAddress, depend directly on StopID, with no transitive dependencies or partial dependencies. The ROUTE table uses RouteID as the primary key and includes foreign keys WarehouseID and DestinationWarehouse. All columns in the ROUTE table depend directly on RouteID and have no transitive dependencies or partial dependencies.

Lastly, the VEHICLE table uses VehicleID as the primary key and includes foreign keys ShipmentID, UserID, and DriverID. All columns, such as Capacity, LicensePlateNumber, and MaintenanceRecords, depend directly on VehicleID, with no transitive dependencies or partial dependencies.

Based on this analysis, all tables in the TMS database schema meet the criteria for 1NF, 2NF, and 3NF. Each table has a primary key, all non-key columns depend solely on the primary key, and there are no partial dependencies or transitive dependencies, ensuring that the database is well-normalized and efficient.

# Implementation and Testing

*Implementation*

The implementation phase involves several key steps to ensure the successful deployment of the database and its integration with the Shiny app on ElephantSQL.

#### *Database Creation*

First, the database schema is created on ElephantSQL. This involves defining all tables, columns, data types, primary keys, and foreign keys as outlined in the provided SQL code. The TMS\_USERS, CUSTOMER, DRIVER, ADMINISTRATOR, DISPATCHER, WAREHOUSE, INVENTORY, ORDER, SHIPMENT, STOP, ROUTE, and VEHICLE tables are created with appropriate constraints to ensure data integrity.

#### *Data Insertion*

Once the schema is established, initial data is inserted into the tables. This includes populating the TMS\_USERS table with user authentication details and roles and adding initial records to other tables such as CUSTOMER, DRIVER, and ORDER. The data insertion process ensures there is enough sample data for testing and demonstration.

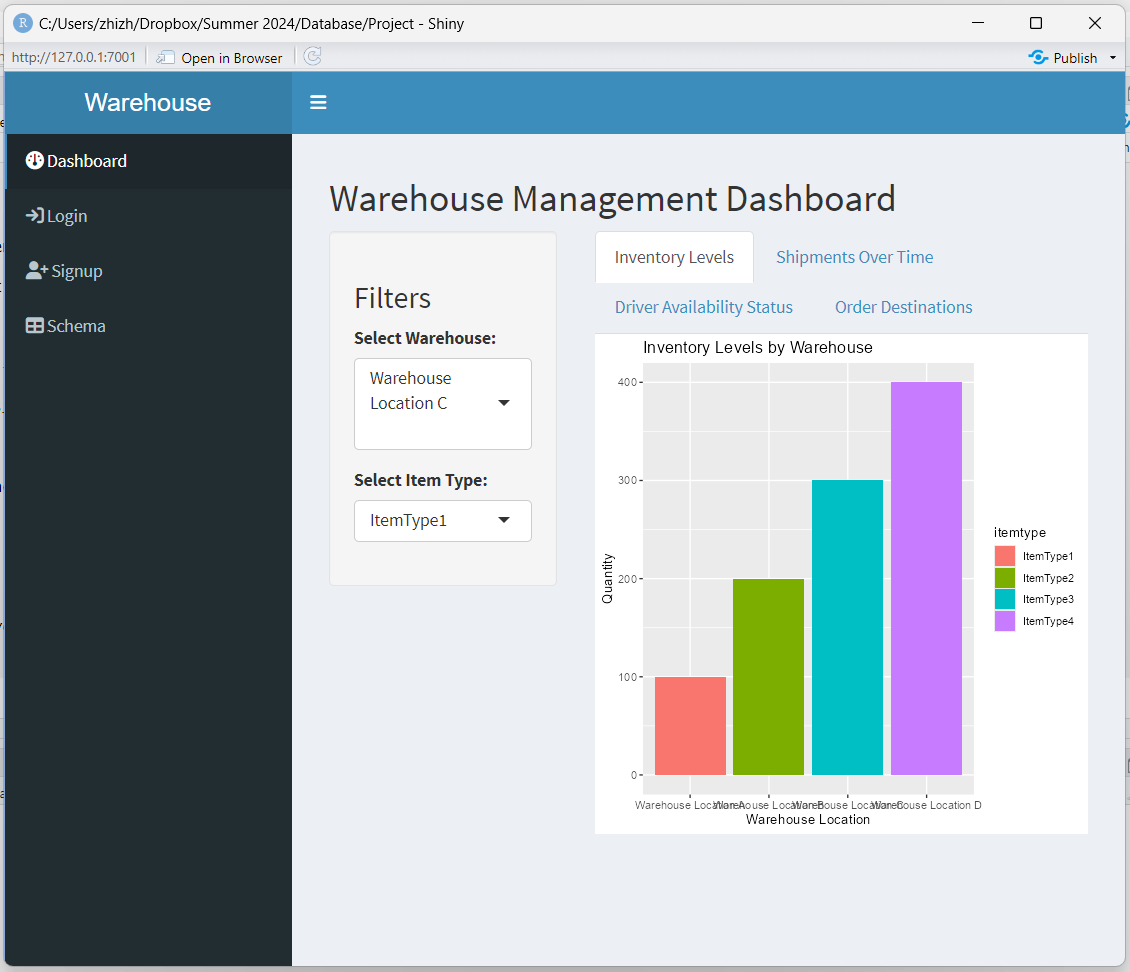
#### *Setting Up Relationships*

Establishing relationships between tables is critical for maintaining referential integrity. Foreign keys are defined to link related tables, such as linking ORDER to CUSTOMER and SHIPMENT to ORDER. These relationships ensure that data is consistently referenced across the database and that operations such as updates and deletions maintain data integrity.

#### *User Interface Integration*

The Shiny app is integrated with the Elephant SQL database to provide a seamless user experience. This involves setting up the database connection within the Shiny app, allowing the app to perform CRUD (Create, Read, Update, Delete) operations on the database. The Shiny app's UI components are designed to interact with the database, enabling users to log in, view data, and generate reports.

This involves setting up the database connection within the Shiny app, allowing the app to perform CRUD (Create, Read, Update, Delete) operations on the database. The Shiny app's UI components are designed to interact with the database, enabling users to log in, view data, and generate reports.



*Screenshot 1*: User-Interface

#### *Data Security*

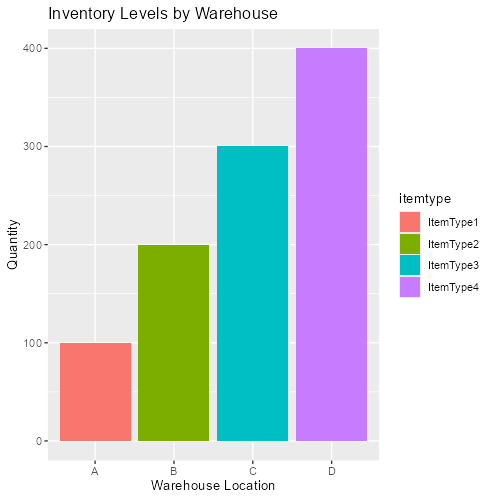
Implementing data security measures is crucial. This includes setting up user authentication to restrict access to the database. Setting up the foundation for implementing role-based access control to ensure that users can only access data relevant to their roles and encryption of sensitive data, such as passwords.

#### *Performance Optimization*

Performance optimization ensures that the database and Shiny app run efficiently. Indexing is applied to frequently queried columns to speed up data retrieval. This includes indexes on columns like Name, Username, EmailAddress in TMS\_USERS, Last\_Login and Created\_Date in CUSTOMER, Location in WAREHOUSE, and other relevant columns in various tables.

#### Reporting & Analytics

The system includes a reporting module capable of generating accurate and timely reports based on the available data. Data visualizations, such as *Figure 1: Inventory Levels by Warehouse* below, are used to present key performance indicators and trends in an easy-to-understand format. These visualizations enhance the comprehension of complex data, facilitating better decision-making. Additional examples of data visualizations are provided in the appendix for further reference.



*Figure 1*: Inventory Levels by Warehouse

## *Testing*

The test plan outlined the strategies and procedures for testing the database functionalities, including user authentication, role-based access control, data integrity, and reporting. Several test cases were developed to ensure comprehensive coverage:

* *Test Case 1:* Verify user authentication with valid and invalid credentials to ensure the system's security.
* *Test Case 2:* Check role-based access control by attempting to access restricted areas with distinct roles, ensuring that users only had access to data relevant to their roles.
* *Test Case 3:* Validate the insertion and retrieval of data from each table, confirming that the system could handle various data operations efficiently.
* *Test Case 4:* Test the accuracy of reports generated by joining multiple tables, ensuring reliable and accurate analytics.
* *Test Case 5:* Ensure foreign key constraints are enforced correctly, maintaining data integrity across related tables.

SQL queries in ElephantSQL were used as the primary testing tools to manually test and validate data insertion, update, and deletion operations.

#### Results

All test cases were successful, demonstrating the robustness and accuracy of the database design and implementation. Specifically, user authentication was verified with both valid and invalid credentials, ensuring that only authorized users could access the system. Role-based access control was successfully enforced, restricting access based on user roles and ensuring data security. Data insertion and retrieval from each table were validated, confirming that the system could handle various data operations efficiently. The accuracy of reports generated by joining multiple tables was confirmed, ensuring that the system could produce reliable and accurate analytics. Additionally, foreign key constraints were correctly enforced, maintaining data integrity across related tables. These successful testing results ensure that the system is reliable and ready for deployment. The SQL DDL and DML scripts, along with the Shiny app code, can be found in the appendix for exploration.

# Conclusions: Challenges & Lessons Learned

Throughout the development of the system, several challenges and valuable lessons emerged. Ensuring data integrity and consistency across multiple related tables was a significant challenge, addressed by implementing primary and foreign key constraints and conducting regular data validation. Securing user authentication and protecting sensitive data were critical issues, resolved by using hashed passwords, role-based access control, and planning for data encryption. Integrating the PostgreSQL database with the Shiny app required a robust database connection setup and thorough testing to ensure smooth operation.

From these experiences, we learned the importance of thorough data validation and integrity checks to maintain system reliability. We also recognized the necessity of robust security measures to protect sensitive information and maintain user trust. Comprehensive testing and incorporating user feedback proved invaluable in identifying issues and improving the system.

# Future Work

Looking ahead, we plan to enhance data security with encryption and regular security audits to ensure data protection. We aim to expand the system's analytics and reporting capabilities to provide deeper insights and better support for decision-making. Improving scalability and performance is also a priority, ensuring the system can handle larger datasets and more concurrent users. Continuously enhancing the user experience through regular updates based on user feedback will be essential for maintaining satisfaction. Additionally, we will explore opportunities to integrate the system with other business systems and third-party services to enhance functionality and provide a more seamless user experience.

References

1. SAP News Center. (2024). "2024 Magic Quadrant for Transportation Management Systems." Retrieved from [SAP News Center](https://news.sap.com).
2. Cleo. (2024). "11 Major Logistics Trends Shaping Logistics Management in 2024." Retrieved from [Cleo](https://www.cleo.com).

# Appendix:

**Data Dictionary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table** | **Attribute** | **Data Type** | **Constraints** | **Description** |
| **TMS\_USERS** | UserID | INT | PRIMARY KEY | Unique identifier for the user |
| Name | VARCHAR(255) | NOT NULL | Full name of the user |
| Username | VARCHAR(255) | NOT NULL UNIQUE | Username for login |
| Password | VARCHAR(255) | NOT NULL | Password for login |
| Address | TEXT |  | Home address of the user |
| EmailAddress | VARCHAR(255) | NOT NULL UNIQUE | Email address of the user |
| PhoneNumber | VARCHAR(20) |  | Phone number of the user |
| Role | VARCHAR(50) | NOT NULL | Role of the user (Customer, Driver, etc.) |
| **CUSTOMER** | UserID | INT | PRIMARY KEY, FOREIGN KEY | References USER(UserID) |
| Last\_Login | DATETIME |  | Last login timestamp |
| Created\_Date | DATETIME | NOT NULL | Account creation date |
| List\_Orders | TEXT |  | List of orders placed by the customer |
| **DRIVER** | UserID | INT | PRIMARY KEY, FOREIGN KEY | References USER(UserID) |
| HourlyWage | DECIMAL(10, 2) | NOT NULL | Hourly wage of the driver |
| LicenseInformation | TEXT | NOT NULL | Driver's license information |
| AvailabilityStatus | VARCHAR(50) | NOT NULL | Availability status of the driver |
| **ADMINISTRATOR** | UserID | INT | PRIMARY KEY, FOREIGN KEY | References USER(UserID) |
| WarehouseID | INT | FOREIGN KEY | References WAREHOUSE(WarehouseID) |
| AnnualSalary | DECIMAL(10, 2) | NOT NULL | Annual salary of the administrator |
| Department | VARCHAR(255) | NOT NULL | Department name |
| ExtensionNumber | VARCHAR(20) |  | Office extension number |
| **DISPATCHER** | UserID | INT | PRIMARY KEY, FOREIGN KEY | References USER(UserID) |
| InventoryID | INT | FOREIGN KEY | References INVENTORY(InventoryID) |
| CommissionRatePerc | DECIMAL(5, 2) | NOT NULL | Commission rate percentage |
| Department | VARCHAR(255) | NOT NULL | Department name |
| ExtensionNumber | VARCHAR(20) |  | Office extension number |
| **WAREHOUSE** | WarehouseID | INT | PRIMARY KEY | Unique identifier for the warehouse |
| Location | TEXT | NOT NULL | Physical location of the warehouse |
| Storage\_Capacity | INT | NOT NULL | Storage capacity of the warehouse |
| **INVENTORY** | InventoryID | INT | PRIMARY KEY | Unique identifier for the inventory item |
| WarehouseID | INT | FOREIGN KEY | References WAREHOUSE(WarehouseID) |
| ItemType | VARCHAR(255) | NOT NULL | Type of item in the inventory |
| Quantity | INT | NOT NULL | Quantity of the item |
| **ORDER** | OrderID | INT | PRIMARY KEY | Unique identifier for the order |
| CustomerID | INT | FOREIGN KEY | References CUSTOMER(UserID) |
| Order\_Date | DATETIME | NOT NULL | Date and time the order was placed |
| Items | TEXT | NOT NULL | Items included in the order |
| Destination\_Address | TEXT | NOT NULL | Delivery address for the order |
| Special\_Instructions | TEXT |  | Any special instructions for the order |
| **SHIPMENT** | ShipmentID | INT | PRIMARY KEY | Unique identifier for the shipment |
| OrderID | INT | FOREIGN KEY | References ORDER(OrderID) |
| InventoryID | INT | FOREIGN KEY | References INVENTORY(InventoryID) |
| Pickup\_Date | DATETIME | NOT NULL | Date and time the shipment was picked up |
| Estimated\_DeliveryDate | DATETIME | NOT NULL | Estimated delivery date and time |
| Current\_Status | VARCHAR(255) | NOT NULL | Current status of the shipment |
| Expense | DECIMAL(10, 2) | NOT NULL | Expense associated with the shipment |
| **STOP** | StopID | INT | PRIMARY KEY | Unique identifier for the stop |
| ShipmentID | INT | FOREIGN KEY | References SHIPMENT(ShipmentID) |
| RouteID | INT | FOREIGN KEY | References ROUTE(RouteID) |
| SequenceNumber | INT | NOT NULL | The sequence number of the stop in the route |
| DeliveryAddress | TEXT | NOT NULL | Delivery address of the stop |
| **ROUTE** | RouteID | INT | PRIMARY KEY | Unique identifier for the route |
| StopID | INT | FOREIGN KEY | References STOP(StopID) |
| WarehouseID | INT | FOREIGN KEY | References WAREHOUSE(WarehouseID) |
| DestinationWarehouse | INT | FOREIGN KEY | References WAREHOUSE(WarehouseID) |
| EstimatedTravelTime | DECIMAL(5, 2) | NOT NULL | Estimated travel time for the route |
| Distance | DECIMAL(10, 2) | NOT NULL | Distance of the route |
| **VEHICLE** | VehicleID | INT | PRIMARY KEY | Unique identifier for the vehicle |
| ShipmentID | INT | FOREIGN KEY | References SHIPMENT(ShipmentID) |
| UserID | INT | FOREIGN KEY | References USER(UserID) |
| VehicleType | VARCHAR(255) | NOT NULL | Type of the vehicle |
| Capacity | INT | NOT NULL | Capacity of the vehicle |
| LicensePlateNumber | VARCHAR(20) | NOT NULL | License plate number of the vehicle |
| MaintenanceRecords | TEXT |  | Maintenance records of the vehicle |
| Driver\_ID | INT | FOREIGN KEY | References DRIVER(UserID) |

**Implementation: SQL DDL and DML Statements**

SQL DDL

-- Create the TMS\_USERS table

CREATE TABLE "TMS\_USERS" (

UserID INT PRIMARY KEY,

Name VARCHAR(255),

Username VARCHAR(255),

Password VARCHAR(255),

Address TEXT,

EmailAddress VARCHAR(255),

PhoneNumber VARCHAR(20),

Role VARCHAR(50)

);

-- Create the CUSTOMER table with a foreign key to the TMS\_USERS table

CREATE TABLE CUSTOMER (

UserID INT PRIMARY KEY,

Last\_Login TIMESTAMP,

Created\_Date TIMESTAMP,

List\_Orders TEXT,

FOREIGN KEY (UserID) REFERENCES "TMS\_USERS"(UserID)

);

-- Create the WAREHOUSE table

CREATE TABLE WAREHOUSE (

WarehouseID INT PRIMARY KEY,

Location TEXT,

Storage\_Capacity INT

);

-- Create the INVENTORY table with a foreign key to the WAREHOUSE table

CREATE TABLE INVENTORY (

InventoryID INT PRIMARY KEY,

WarehouseID INT,

ItemType VARCHAR(255),

Quantity INT,

FOREIGN KEY (WarehouseID) REFERENCES WAREHOUSE(WarehouseID)

);

-- Create the ADMINISTRATOR table with foreign keys to the TMS\_USERS and WAREHOUSE tables

CREATE TABLE ADMINISTRATOR (

UserID INT PRIMARY KEY,

WarehouseID INT,

AnnualSalary DECIMAL(10, 2),

Department VARCHAR(255),

ExtensionNumber VARCHAR(20),

FOREIGN KEY (UserID) REFERENCES "TMS\_USERS"(UserID),

FOREIGN KEY (WarehouseID) REFERENCES WAREHOUSE(WarehouseID)

);

-- Create the DRIVER table with a foreign key to the TMS\_USERS table

CREATE TABLE DRIVER (

UserID INT PRIMARY KEY,

HourlyWage DECIMAL(10, 2),

LicenseInformation TEXT,

AvailabilityStatus VARCHAR(50),

FOREIGN KEY (UserID) REFERENCES "TMS\_USERS"(UserID)

);

-- Create the DISPATCHER table with foreign keys to the TMS\_USERS and INVENTORY tables

CREATE TABLE DISPATCHER (

UserID INT PRIMARY KEY,

InventoryID INT,

CommissionRatePerc DECIMAL(5, 2),

Department VARCHAR(255),

ExtensionNumber VARCHAR(20),

FOREIGN KEY (UserID) REFERENCES "TMS\_USERS"(UserID),

FOREIGN KEY (InventoryID) REFERENCES INVENTORY(InventoryID)

);

-- Create the ORDER table with a foreign key to the CUSTOMER table

CREATE TABLE "ORDER" (

OrderID INT PRIMARY KEY,

CustomerID INT,

Order\_Date TIMESTAMP,

Items TEXT,

Destination\_Address TEXT,

Special\_Instructions TEXT,

FOREIGN KEY (CustomerID) REFERENCES CUSTOMER(UserID)

);

-- Create the SHIPMENT table with foreign keys to the ORDER and INVENTORY tables

CREATE TABLE SHIPMENT (

ShipmentID INT PRIMARY KEY,

OrderID INT,

InventoryID INT,

Pickup\_Date TIMESTAMP,

Estimated\_DeliveryDate TIMESTAMP,

Current\_Status VARCHAR(255),

Expense DECIMAL(10, 2),

FOREIGN KEY (OrderID) REFERENCES "ORDER"(OrderID),

FOREIGN KEY (InventoryID) REFERENCES INVENTORY(InventoryID)

);

-- Create the STOP table with foreign keys to the SHIPMENT and ROUTE tables

CREATE TABLE STOP (

StopID INT PRIMARY KEY,

ShipmentID INT,

RouteID INT,

SequenceNumber INT,

DeliveryAddress TEXT,

FOREIGN KEY (ShipmentID) REFERENCES SHIPMENT(ShipmentID)

);

-- Create the ROUTE table with foreign keys to the STOP and WAREHOUSE tables

CREATE TABLE ROUTE (

RouteID INT PRIMARY KEY,

StopID INT,

WarehouseID INT,

DestinationWarehouse INT,

EstimatedTravelTime DECIMAL(5, 2),

Distance DECIMAL(10, 2),

FOREIGN KEY (StopID) REFERENCES STOP(StopID),

FOREIGN KEY (WarehouseID) REFERENCES WAREHOUSE(WarehouseID)

);

-- Create the VEHICLE table with foreign keys to the SHIPMENT, TMS\_USERS, and DRIVER tables

CREATE TABLE VEHICLE (

VehicleID INT PRIMARY KEY,

ShipmentID INT,

UserID INT,

VehicleType VARCHAR(255),

Capacity INT,

LicensePlateNumber VARCHAR(20),

MaintenanceRecords TEXT,

Driver\_ID INT,

FOREIGN KEY (ShipmentID) REFERENCES SHIPMENT(ShipmentID),

FOREIGN KEY (UserID) REFERENCES "TMS\_USERS"(UserID),

FOREIGN KEY (Driver\_ID) REFERENCES DRIVER(UserID)

);

SQL DML Statements

-- Insert data into TMS\_USERS table

INSERT INTO "TMS\_USERS" (UserID, Name, Username, Password, Address, EmailAddress, PhoneNumber, Role) VALUES

(1, 'John Doe', 'jdoe', 'password123', '123 Main St', '[jdoe@example.com](mailto:jdoe@example.com)', '555-1234', 'Customer'),

(2, 'Jane Smith', 'jsmith', 'password456', '456 Elm St', '[jsmith@example.com](mailto:jsmith@example.com)', '555-5678', 'Driver'),

(3, 'Bob Johnson', 'bjohnson', 'password789', '789 Pine St', '[bjohnson@example.com](mailto:bjohnson@example.com)', '555-9012', 'Administrator'),

(4, 'Alice Williams', 'awilliams', 'password012', '012 Oak St', '[awilliams@example.com](mailto:awilliams@example.com)', '555-3456', 'Dispatcher'),

(5, 'Charlie Brown', 'cbrown', 'password321', '654 Maple St', '[cbrown@example.com](mailto:cbrown@example.com)', '555-6789', 'Customer'),

(6, 'Lucy Van Pelt', 'lvanpelt', 'password654', '321 Cedar St', '[lvanpelt@example.com](mailto:lvanpelt@example.com)', '555-2345', 'Driver'),

(7, 'Linus Van Pelt', 'lvanpelt2', 'password987', '987 Willow St', '[lvanpelt2@example.com](mailto:lvanpelt2@example.com)', '555-6780', 'Administrator'),

(8, 'Sally Brown', 'sbrown', 'password210', '432 Birch St', '[sbrown@example.com](mailto:sbrown@example.com)', '555-3457', 'Dispatcher');

-- Insert data into CUSTOMER table

INSERT INTO CUSTOMER (UserID, Last\_Login, Created\_Date, List\_Orders) VALUES

(5, '2024-06-15 09:00:00', '2024-01-02 11:00:00', 'Order3, Order4'),

(1, '2024-06-15 10:00:00', '2024-01-01 12:00:00', 'Order5, Order6');

-- Insert data into WAREHOUSE table

INSERT INTO WAREHOUSE (WarehouseID, Location, Storage\_Capacity) VALUES

(1, 'Warehouse Location A', 1000),

(2, 'Warehouse Location B', 2000),

(3, 'Warehouse Location C', 3000),

(4, 'Warehouse Location D', 4000);

-- Insert data into INVENTORY table

INSERT INTO INVENTORY (InventoryID, WarehouseID, ItemType, Quantity) VALUES

(1, 1, 'ItemType1', 100),

(2, 2, 'ItemType2', 200),

(3, 3, 'ItemType3', 300),

(4, 4, 'ItemType4', 400);

-- Insert data into ADMINISTRATOR table

INSERT INTO ADMINISTRATOR (UserID, WarehouseID, AnnualSalary, Department, ExtensionNumber) VALUES

(7, 3, 80000, 'Admin Dept 2', '102'),

(3, 4, 90000, 'Admin Dept 3', '103');

-- Insert data into DRIVER table

INSERT INTO DRIVER (UserID, HourlyWage, LicenseInformation, AvailabilityStatus) VALUES

(6, 22.00, 'License456', 'Unavailable'),

(2, 25.00, 'License789', 'Available');

-- Insert data into DISPATCHER table

INSERT INTO DISPATCHER (UserID, InventoryID, CommissionRatePerc, Department, ExtensionNumber) VALUES

(8, 3, 6.00, 'Dispatch Dept 2', '203'),

(4, 4, 7.00, 'Dispatch Dept 3', '204');

-- Insert data into ORDER table

INSERT INTO "ORDER" (OrderID, CustomerID, Order\_Date, Items, Destination\_Address, Special\_Instructions) VALUES

(2, 5, '2024-06-11 13:00:00', 'Item3, Item4', '123 Spruce St', 'Leave at the back door'),

(3, 1, '2024-06-12 14:00:00', 'Item5, Item6', '456 Fir St', 'Ring the bell');

-- Insert data into SHIPMENT table

INSERT INTO SHIPMENT (ShipmentID, OrderID, InventoryID, Pickup\_Date, Estimated\_DeliveryDate, Current\_Status, Expense) VALUES

(2, 2, 3, '2024-06-13 10:00:00', '2024-06-14 10:00:00', 'Delivered', 60.00),

(3, 3, 4, '2024-06-14 11:00:00', '2024-06-15 11:00:00', 'Pending', 70.00);

-- Insert data into STOP table

INSERT INTO STOP (StopID, ShipmentID, RouteID, SequenceNumber, DeliveryAddress) VALUES

(2, 2, 2, 1, '123 Spruce St'),

(3, 3, 3, 1, '456 Fir St');

-- Insert data into ROUTE table

INSERT INTO ROUTE (RouteID, StopID, WarehouseID, DestinationWarehouse, EstimatedTravelTime, Distance) VALUES

(2, 2, 2, 3, 3.0, 200),

(3, 3, 3, 4, 4.0, 250);

-- Insert data into VEHICLE table

INSERT INTO VEHICLE (VehicleID, ShipmentID, UserID, VehicleType, Capacity, LicensePlateNumber, MaintenanceRecords, Driver\_ID) VALUES

(2, 2, 6, 'Van', 1200, 'DEF456', 'Minor Maintenance', 6),

(3, 3, 2, 'Bike', 500, 'GHI789', 'No Maintenance', 2);

-- Create indexes for TMS\_USERS table

CREATE INDEX idx\_tms\_users\_name ON "TMS\_USERS"(Name);

CREATE INDEX idx\_tms\_users\_username ON "TMS\_USERS"(Username);

CREATE INDEX idx\_tms\_users\_email ON "TMS\_USERS"(EmailAddress);

-- Create indexes for CUSTOMER table

CREATE INDEX idx\_customer\_last\_login ON CUSTOMER(Last\_Login);

CREATE INDEX idx\_customer\_created\_date ON CUSTOMER(Created\_Date);

-- Create indexes for WAREHOUSE table

CREATE INDEX idx\_warehouse\_location ON WAREHOUSE(Location);

-- Create indexes for INVENTORY table

CREATE INDEX idx\_inventory\_warehouse\_id ON INVENTORY(WarehouseID);

CREATE INDEX idx\_inventory\_item\_type ON INVENTORY(ItemType);

-- Create indexes for ADMINISTRATOR table

CREATE INDEX idx\_administrator\_warehouse\_id ON ADMINISTRATOR(WarehouseID);

CREATE INDEX idx\_administrator\_department ON ADMINISTRATOR(Department);

-- Create indexes for DRIVER table

CREATE INDEX idx\_driver\_availability\_status ON DRIVER(AvailabilityStatus);

CREATE INDEX idx\_driver\_hourly\_wage ON DRIVER(HourlyWage);

-- Create indexes for DISPATCHER table

CREATE INDEX idx\_dispatcher\_inventory\_id ON DISPATCHER(InventoryID);

CREATE INDEX idx\_dispatcher\_department ON DISPATCHER(Department);

-- Create indexes for ORDER table

CREATE INDEX idx\_order\_customer\_id ON "ORDER"(CustomerID);

CREATE INDEX idx\_order\_order\_date ON "ORDER"(Order\_Date);

-- Create indexes for SHIPMENT table

CREATE INDEX idx\_shipment\_order\_id ON SHIPMENT(OrderID);

CREATE INDEX idx\_shipment\_inventory\_id ON SHIPMENT(InventoryID);

CREATE INDEX idx\_shipment\_pickup\_date ON SHIPMENT(Pickup\_Date);

-- Create indexes for STOP table

CREATE INDEX idx\_stop\_shipment\_id ON STOP(ShipmentID);

CREATE INDEX idx\_stop\_route\_id ON STOP(RouteID);

-- Create indexes for ROUTE table

CREATE INDEX idx\_route\_warehouse\_id ON ROUTE(WarehouseID);

CREATE INDEX idx\_route\_destination\_warehouse ON ROUTE(DestinationWarehouse);

-- Create indexes for VEHICLE table

CREATE INDEX idx\_vehicle\_shipment\_id ON VEHICLE(ShipmentID);

CREATE INDEX idx\_vehicle\_user\_id ON VEHICLE(UserID);

CREATE INDEX idx\_vehicle\_driver\_id ON VEHICLE(Driver\_ID);

CREATE INDEX idx\_vehicle\_vehicle\_type ON VEHICLE(VehicleType);

-- Update a user's email address

UPDATE "TMS\_USERS"

SET EmailAddress = '[john.doe@newdomain.com](mailto:john.doe@newdomain.com)'

WHERE UserID = 1;

-- Update a driver's availability status

UPDATE DRIVER

SET AvailabilityStatus = 'Unavailable'

WHERE UserID = 2;

-- Update a warehouse's storage capacity

UPDATE WAREHOUSE

SET Storage\_Capacity = 1500

WHERE WarehouseID = 1;

-- Update an order's special instructions

UPDATE "ORDER"

SET Special\_Instructions = 'Leave at the side door'

WHERE OrderID = 1;

-- Delete a user

DELETE FROM "TMS\_USERS"

WHERE UserID = 5;

-- Delete an order

DELETE FROM "ORDER"

WHERE OrderID = 2;

-- Delete a shipment

DELETE FROM SHIPMENT

WHERE ShipmentID = 2;

-- Delete an inventory item

DELETE FROM INVENTORY

WHERE InventoryID = 3;

-- Select all users

SELECT \* FROM "TMS\_USERS";

-- Select all drivers and their availability status

SELECT "TMS\_USERS".Name, DRIVER.AvailabilityStatus

FROM "TMS\_USERS"

JOIN DRIVER ON "TMS\_USERS".UserID = DRIVER.UserID;

-- Select all orders placed by a specific customer

SELECT \* FROM "ORDER"

WHERE CustomerID = 1;

-- Select all shipments with their current status

SELECT SHIPMENT.ShipmentID, "ORDER".OrderID, SHIPMENT.Current\_Status

FROM SHIPMENT

JOIN "ORDER" ON SHIPMENT.OrderID = "ORDER".OrderID;

-- Select inventory items and their quantities in a specific warehouse

SELECT INVENTORY.ItemType, INVENTORY.Quantity

FROM INVENTORY

WHERE WarehouseID = 1;

-- Select all routes and their estimated travel times

SELECT \* FROM ROUTE;

-- Select all vehicles and their driver details

SELECT VEHICLE.VehicleID, "TMS\_USERS".Name AS DriverName, VEHICLE.VehicleType, VEHICLE.LicensePlateNumber

FROM VEHICLE

JOIN "TMS\_USERS" ON VEHICLE.Driver\_ID = "TMS\_USERS".UserID;

-- Select all administrators and the warehouses they manage

SELECT "TMS\_USERS".Name AS AdminName, WAREHOUSE.Location AS WarehouseLocation

FROM ADMINISTRATOR

JOIN "TMS\_USERS" ON ADMINISTRATOR.UserID = "TMS\_USERS".UserID

JOIN WAREHOUSE ON ADMINISTRATOR.WarehouseID = WAREHOUSE.WarehouseID;

-- Select all users who are drivers

SELECT "TMS\_USERS".UserID, "TMS\_USERS".Name, "TMS\_USERS".EmailAddress, DRIVER.HourlyWage

FROM "TMS\_USERS"

JOIN DRIVER ON "TMS\_USERS".UserID = DRIVER.UserID;

-- Select all orders placed by a specific customer with order details

SELECT "ORDER".OrderID, "ORDER".Order\_Date, "ORDER".Items, "ORDER".Destination\_Address, CUSTOMER.UserID, CUSTOMER.Last\_Login

FROM "ORDER"

JOIN CUSTOMER ON "ORDER".CustomerID = CUSTOMER.UserID

WHERE CUSTOMER.UserID = 1;

-- Select all shipments and their statuses along with order and customer details

SELECT SHIPMENT.ShipmentID, SHIPMENT.Current\_Status, "ORDER".OrderID, "ORDER".Order\_Date, "TMS\_USERS".Name AS CustomerName, "TMS\_USERS".EmailAddress

FROM SHIPMENT

JOIN "ORDER" ON SHIPMENT.OrderID = "ORDER".OrderID

JOIN CUSTOMER ON "ORDER".CustomerID = CUSTOMER.UserID

JOIN "TMS\_USERS" ON CUSTOMER.UserID = "TMS\_USERS".UserID;

-- Select total quantity of each item type in inventory across all warehouses

SELECT ItemType, SUM(Quantity) AS TotalQuantity

FROM INVENTORY

GROUP BY ItemType;

-- Select the average hourly wage of drivers

SELECT AVG(HourlyWage) AS AverageHourlyWage

FROM DRIVER;

-- Select all stops along with their shipment and route details

SELECT STOP.StopID, STOP.SequenceNumber, STOP.DeliveryAddress, SHIPMENT.ShipmentID, ROUTE.RouteID, ROUTE.EstimatedTravelTime

FROM STOP

JOIN SHIPMENT ON STOP.ShipmentID = SHIPMENT.ShipmentID

JOIN ROUTE ON STOP.RouteID = ROUTE.RouteID;

-- Select all vehicles and their associated driver details

SELECT VEHICLE.VehicleID, VEHICLE.VehicleType, VEHICLE.LicensePlateNumber, DRIVER.UserID AS DriverID, "TMS\_USERS".Name AS DriverName

FROM VEHICLE

JOIN DRIVER ON VEHICLE.Driver\_ID = DRIVER.UserID

JOIN "TMS\_USERS" ON DRIVER.UserID = "TMS\_USERS".UserID;

-- Select all administrators and the warehouses they manage along with warehouse details

SELECT ADMINISTRATOR.UserID AS AdminID, "TMS\_USERS".Name AS AdminName, WAREHOUSE.WarehouseID, WAREHOUSE.Location

FROM ADMINISTRATOR

JOIN "TMS\_USERS" ON ADMINISTRATOR.UserID = "TMS\_USERS".UserID

JOIN WAREHOUSE ON ADMINISTRATOR.WarehouseID = WAREHOUSE.WarehouseID;

-- Select the total number of orders placed by each customer

SELECT CUSTOMER.UserID, "TMS\_USERS".Name, COUNT("ORDER".OrderID) AS TotalOrders

FROM CUSTOMER

JOIN "ORDER" ON CUSTOMER.UserID = "ORDER".CustomerID

JOIN "TMS\_USERS" ON CUSTOMER.UserID = "TMS\_USERS".UserID

GROUP BY CUSTOMER.UserID, "TMS\_USERS".Name;

-- Select all shipments that are still in transit and their details

SELECT SHIPMENT.ShipmentID, SHIPMENT.Pickup\_Date, SHIPMENT.Estimated\_DeliveryDate, SHIPMENT.Current\_Status, "ORDER".OrderID, "ORDER".Items

FROM SHIPMENT

JOIN "ORDER" ON SHIPMENT.OrderID = "ORDER".OrderID

WHERE SHIPMENT.Current\_Status = 'In Transit';

-- Select the route with the longest estimated travel time

SELECT RouteID, MAX(EstimatedTravelTime) AS LongestTravelTime

FROM ROUTE

GROUP BY RouteID

ORDER BY LongestTravelTime DESC

LIMIT 1;

-- Select the driver with the highest hourly wage

SELECT "TMS\_USERS".UserID, "TMS\_USERS".Name, DRIVER.HourlyWage

FROM DRIVER

JOIN "TMS\_USERS" ON DRIVER.UserID = "TMS\_USERS".UserID

ORDER BY DRIVER.HourlyWage DESC

LIMIT 1;

-- Select the average expense per shipment

SELECT AVG(Expense) AS AverageExpense

FROM SHIPMENT;

**Shiny APP**

library(shiny)

library(shinydashboard)

library(shinyjs)

# Define UI for application

ui <- dashboardPage(

dashboardHeader(title = "Warehouse Management Dashboard"),

dashboardSidebar(

sidebarMenu(

menuItem("Dashboard", tabName = "dashboard", icon = icon("dashboard")),

menuItem("Login", tabName = "login", icon = icon("sign-in")),

menuItem("Signup", tabName = "signup", icon = icon("user-plus")),

menuItem("Schema", tabName = "schema", icon = icon("table"))

)

),

dashboardBody(

useShinyjs(),

tabItems(

tabItem(tabName = "dashboard",

fluidPage(

h2("Warehouse Management Dashboard"),

sidebarLayout(

sidebarPanel(

h3("Filters"),

selectInput("warehouse", "Select Warehouse:", choices = NULL),

selectInput("itemtype", "Select Item Type:", choices = NULL)

),

mainPanel(

tabsetPanel(

tabPanel("Inventory Levels", plotOutput("inventoryPlot")),

tabPanel("Shipments Over Time", plotOutput("shipmentsPlot")),

tabPanel("Driver Availability Status", plotOutput("driverStatusPlot")),

tabPanel("Order Destinations", plotOutput("orderDestinationsPlot"))

)

)

)

)),

tabItem(tabName = "login",

fluidPage(

h2("Login"),

textInput("login\_username", "Username"),

passwordInput("login\_password", "Password"),

actionButton("login\_button", "Login"),

textOutput("login\_status")

)),

tabItem(tabName = "signup",

fluidPage(

h2("Signup"),

textInput("signup\_username", "Username"),

passwordInput("signup\_password", "Password"),

actionButton("signup\_button", "Signup"),

textOutput("signup\_status")

)),

tabItem(tabName = "schema",

fluidPage(

h2("Database Schema"),

DT::dataTableOutput("schemaTable")

))

)

)

)

# Load necessary libraries

library(DBI)

library(RPostgres)

library(dplyr)

library(ggplot2)

library(DT)

# Establish a connection to the PostgreSQL database using environment variables

con <- dbConnect(RPostgres::Postgres(),

dbname = Sys.getenv('DB\_NAME'),

host = Sys.getenv('DB\_HOST'),

port = as.integer(Sys.getenv('DB\_PORT')),

user = Sys.getenv('DB\_USER'),

password = Sys.getenv('DB\_PASSWORD'))

# Function to check credentials against the tms\_users table in the database

check\_credentials <- function(username, password) {

query <- paste0("SELECT \* FROM \"TMS\_USERS\" WHERE \"username\" = '", username, "' AND \"password\" = '", password, "'")

user <- dbGetQuery(con, query)

return(nrow(user) == 1)

}

# Function to add new users to the tms\_users table in the database

add\_user <- function(username, password) {

# Generate a unique userid

max\_userid\_query <- dbGetQuery(con, "SELECT COALESCE(MAX(\"userid\"), 0) as max\_userid FROM \"TMS\_USERS\"")

userid <- as.integer(max\_userid\_query$max\_userid) + 1

query <- paste0("INSERT INTO \"TMS\_USERS\" (\"userid\", \"username\", \"password\", \"role\") VALUES (", userid, ", '", username, "', '", password, "', 'Customer')")

dbExecute(con, query)

}

# Function to get data from the database

get\_data <- function(query) {

dbGetQuery(con, query)

}

# Function to create plots

create\_plot <- function(data, x, y, title, xlab, ylab, geom\_type = "bar", fill = NULL, position = "dodge") {

p <- ggplot(data, aes\_string(x = x, y = y, fill = fill))

if (geom\_type == "bar") {

p <- p + geom\_bar(stat = "identity", position = position)

} else if (geom\_type == "line") {

p <- p + geom\_line()

} else if (geom\_type == "pie") {

p <- p + geom\_bar(stat = "identity", width = 1) + coord\_polar("y")

}

p + labs(title = title, x = xlab, y = ylab)

}

# Function to create the inventory plot

create\_inventory\_plot <- function() {

inventory\_levels <- get\_data("

SELECT w.\"location\" AS location, i.\"itemtype\" AS itemtype, i.\"quantity\" AS quantity

FROM \"inventory\" i

JOIN \"warehouse\" w ON i.\"warehouseid\" = w.\"warehouseid\"

")

# Create a mapping of full location names to shorter labels

location\_mapping <- c("Warehouse Location A" = "A",

"Warehouse Location B" = "B",

"Warehouse Location C" = "C",

"Warehouse Location D" = "D")

# Apply the mapping to the location column

inventory\_levels$location <- location\_mapping[inventory\_levels$location]

create\_plot(inventory\_levels, "location", "quantity", "Inventory Levels by Warehouse", "Warehouse Location", "Quantity", fill = "itemtype")

}

# Function to create the shipments over time plot

create\_shipments\_plot <- function() {

shipments\_over\_time <- get\_data("

SELECT \"pickup\_date\" AS pickup\_date, COUNT(\*) as shipment\_count

FROM \"shipment\"

GROUP BY \"pickup\_date\"

ORDER BY \"pickup\_date\"

")

create\_plot(shipments\_over\_time, "as.Date(pickup\_date)", "shipment\_count", "Shipments Over Time", "Date", "Number of Shipments", geom\_type = "line")

}

# Function to create the driver availability status plot

create\_driver\_status\_plot <- function() {

driver\_status <- get\_data("

SELECT \"availabilitystatus\" AS availability\_status, COUNT(\*) as driver\_count

FROM \"driver\"

GROUP BY \"availabilitystatus\"

")

if (nrow(driver\_status) == 0) {

return(NULL)

}

create\_plot(driver\_status, "availability\_status", "driver\_count", "Driver Availability Status", NULL, "Number of Drivers", geom\_type = "pie", fill = "availability\_status")

}

# Function to create the order destinations plot

create\_order\_destinations\_plot <- function() {

order\_destinations <- get\_data("

SELECT \"destination\_address\" AS destination\_address, COUNT(\*) as order\_count

FROM \"ORDER\"

GROUP BY \"destination\_address\"

")

create\_plot(order\_destinations, "reorder(destination\_address, -order\_count)", "order\_count", "Order Destinations", "Destination Address", "Number of Orders") + coord\_flip()

}

# Function to get table and column information

get\_table\_columns <- function() {

query <- "

SELECT table\_name, column\_name, data\_type

FROM information\_schema.columns

WHERE table\_schema = 'public'

ORDER BY table\_name, ordinal\_position;

"

dbGetQuery(con, query)

}

# Define server logic

server <- function(input, output, session) {

# Reactive value to store login status

user\_logged\_in <- reactiveVal(FALSE)

observeEvent(input$login\_button, {

username <- input$login\_username

password <- input$login\_password

if (check\_credentials(username, password)) {

shinyjs::alert("Login successful!")

user\_logged\_in(TRUE)

updateTabItems(session, "tabs", "dashboard")

} else {

output$login\_status <- renderText("Invalid username or password!")

}

})

observeEvent(input$signup\_button, {

username <- input$signup\_username

password <- input$signup\_password

if (username != "" & password != "") {

tryCatch({

add\_user(username, password)

output$signup\_status <- renderText("Signup successful! Please login.")

}, error = function(e) {

output$signup\_status <- renderText(paste("Signup failed! Error: ", e$message))

})

} else {

output$signup\_status <- renderText("Both fields are required!")

}

})

observe({

if (user\_logged\_in()) {

warehouses <- get\_data("SELECT DISTINCT \"location\" FROM \"warehouse\"")

itemtypes <- get\_data("SELECT DISTINCT \"itemtype\" FROM \"inventory\"")

updateSelectInput(session, "warehouse", choices = warehouses$location)

updateSelectInput(session, "itemtype", choices = itemtypes$itemtype)

}

})

output$inventoryPlot <- renderPlot({

if (user\_logged\_in()) {

create\_inventory\_plot()

}

})

output$shipmentsPlot <- renderPlot({

if (user\_logged\_in()) {

create\_shipments\_plot()

}

})

output$driverStatusPlot <- renderPlot({

if (user\_logged\_in()) {

create\_driver\_status\_plot()

}

})

output$orderDestinationsPlot <- renderPlot({

if (user\_logged\_in()) {

create\_order\_destinations\_plot()

}

})

output$schemaTable <- DT::renderDataTable({

get\_table\_columns()

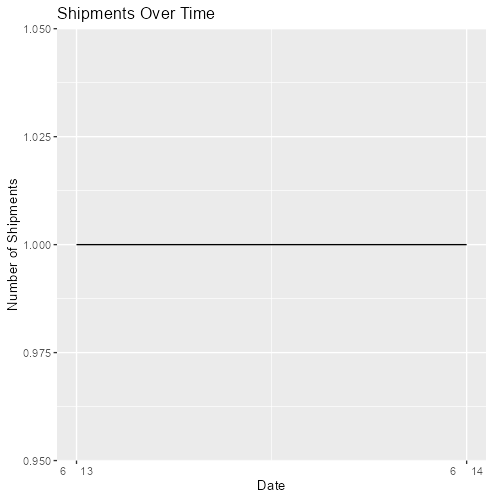
})

}

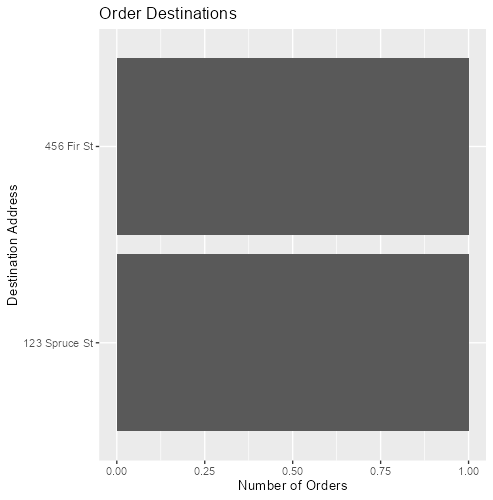
# Run the application

shinyApp(ui = ui, server = server)

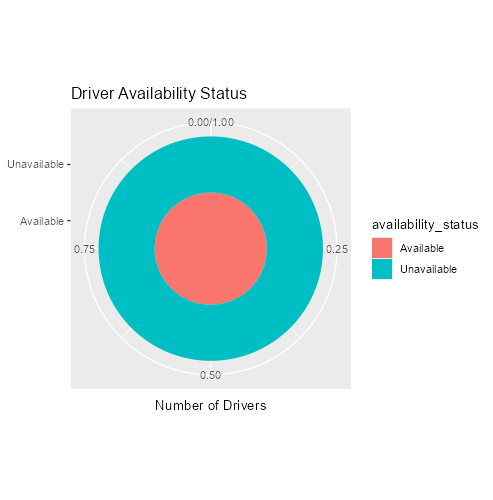
REPORTING METRICS



*Figure 2*: Shipment Over Time



*Figure 3*: Order Destinations.



*Figure 4*: Driver Availability.