DUTCHESS COUNTY BUS TRANSPORTATION SYSTEM

MSCS_542L_256_23S

The Four-Ce

Marist College School of Computer Science and Mathematics

Submitted To: Dr. Reza Sadeghi

11-22-2023

PROJECT REPORT OF DUTCHESS COUNTY BUS TRANSPORTATION SYSTEM

Team Name

The Four-Ce

Team Members

1. Sindhuja Ravikanth (Team Head)	Sindhuja.Ravikanth1@marist.edu
2. Shanmukha Chowdary Nalla (Teammember)	ShanumukhaChowdary.Nalla1@marist.edu
3. Gaurav Bapurao Sherla (Team member)	GauravBapurao.Sherla1@marist.edu
4. Katipally Chanakya Vardhan Reddy (Teammember)	ChanakyaVardhanReddy.Katipally1@marist.edu

TABLE OF CONTENTS

TABLE OF FIGURES

DESCRIPTION OF TEAM MEMBERS

1. **Sindhuja Ravikanth**

   I attained my bachelor's degree in computer science from Keshav Memorial Institute of Technology in 2022. I worked as a Software Developer in TCP Wave for 1 year 9 months. I have come to Marist College for my master's degree in computer science. I am proficient in Java, MySQL, and JavaScript.I am here to develop my skills and develop better products.

2. **Shanmukha Chowdary Nalla**

   I graduated from the Indian Institute of Technology Design and Manufacturing, Jabalpur in the year 2022. I started working in the software field in the 7th semester at Merkle Company as an Analyst and then as a Salesforce Marketing Cloud Developer till June 2023. I am a CertifiedSalesforce Marketing Cloud Developer, Email Specialist, and Machine Learning Engineer. I chose this course because it helps me enhance my skillsetin data management which is always handy.

3. **Gaurav Bapurao Sherla**

   I am an aspiring AWS and ML developer. I want to make the world a betterplace by empowering technology with the help of AI, especially in the mining sector. Where many lives are lost while mining ore from hundreds of feet beneath the earth, I have worked as an ML developer for a startup where I gained some useful insights with the help of this course I will be able to get in-depth practical knowledge about DBMS and this would help me enhance my knowledge, which can help me boost my skills for my futuregrowth.

4. **Katipally Chanakya Vardhan Reddy**

   I am Katipally Chanakya Vardhan Reddy, a Computer Science graduate from Methodist College of Engineering and Technology. Currently, I'm pursuing my Master's in MSCS with a focus on cloud computing at MaristCollege. I excel in Python, C, and C++ and have a strong background in problem-solving.

1.INTRODUCTION

In an era defined by rapid urbanization and increased reliance on public transportation, the Dutchess County Bus Transportation System (DCBTS) stands as a vital lifeline for the residents of Dutchess County, New York. TheDCBTS is an essential part of the daily lives of countless commuters, offeringa means to connect with their destinations efficiently and sustainably. However, in a world where technological advancements are reshaping how we interact with the world around us, it is imperative that our public transportation system evolves in tandem.

The objective of this project Is to propel the DCBTS Into the future by revolutionizing its digital presence and user experience. With the advent of the Information Age, accessibility to transportation information is crucial, andthe current DCBTS app, while functional, requires a comprehensive overhaulto meet the needs and expectations of modern commuters.

The primary aim of this project is to develop a user-friendly DCBTS application that caters to a diverse range of users. This includes commuters seeking to efficiently plan their journeys, tourists exploring the county, and an administrator with specialized privileges to manage and optimize the system.

In this project, we are planning to simplify the process of planning daily commutes via DCPT buses for all individuals which would be user-friendly to the users of all the age groups above 10 years. With the help of our projectmodel, we want to cater our services to our users by providing them with enough data in a simplified format. Our user interface will provide the users with a menu where they can put in their source and destination locations andall the bus routes and buses in respect to their search will be displayed. Whichcan help them keep track of their bus timings and changes of any kind in thebus schedule. The users can also mark their favorite routes and buses which they mostly use for their commute and can keep track of them without any inconvenience.

2.PROJECT OBJECTIVES

The following should be supported by the system:

- The application should be able to prompt the commuters for the login detailsand allow them to change credentials with proper authentication.
- The application should allow the administrator to make changes in the user records.
- The application should provide the capability for the users or administrator to modify their contact information and profile changes with proper authentication.
- Users should be able to access information about their bus routes and buses atany point in time.
- The administrator should have access to the user's credentials through a valid process.
- The administrator should be able to make changes in the commuting-related data.
- The application should have the capability to access separate login databaseslike commuter login and administrator login. So that they can access their respective data and use the application based on their requirements.

3.REVIEW

3.1 Dutchess County Public Transportation (DCPT) [1]

- This app is based on the whole public transportation system of Dutchess County.
- The app shows the live tracking of buses around Dutchess County.
- The buses are identified by their bus numbers with which people can identify which bus would be comfortable to board based on their schedule.
- The app also shows the user's live location, so that the user can identify whichbus is the nearest to them.
- This app shows buses in a map form.

3.2 NYC Transit: MTA Subway & Bus [2]

- This app is based on the Subways and Buses of New York.
- This app shows the best ways to get from a source to a destination around New York.
- It saves different locations as favorites.

3.3 Moovit [3]

- This app is based on all types of transport in many countries worldwide.
- It shows directions from one place to another and suggests quicker ways toget from a source to a destination irrespective of the transit type.

4.MERITS

- The DCBTS helps everyone to search for a route from source to destinationand plan their day based on the time estimation of the buses in Dutchess County.
- This application shows all the stops between the source and destination which is not available in any of the reviewed applications.
- This application will keep track of days when the buses will not be availablelike public holidays which is not available in any of the reviewed applications.
- This application will support multiple users including an Administrator.
- This application will also show the bus fare by taking into consideration theuser's age which is not available in any of the reviewed applications.

5.GITHUB REPOSITORY

https://github.com/SindhujaRavikanth2001/DBMS_Project

6.ENTITY RELATIONSHIP MODEL (ER MODEL)

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

We have used ERDPlus to create the ER diagram. ERDPlus is an open-source software modeling tool that supports the UML (Unified Modeling Language) framework for system and software modeling.

Entities	**Attributes**
User	UserID, FirstName, LastName, DOB, Email, PhoneNumber, Username, Password
Admin	AdminID, Username, Password, Email, PhoneNumber, EmployeeID
Bus	BusID, BusNumber, Capacity,LicensePlateNumber, BusType
Ticket Type	TicketTypeID, TypeName, Price, ValidityPeriod, Description, UserID
Payment	PaymentID, Amount, UserID, TicketTypeID, PaymentDate
Reservation	ReservationID, PaymentID, BusID, ReservationDate, NumberOfReservations
Bus Route	RouteID, StartLocation, EndLocation, BusID
Route Stop Sequence	StopSequenceID, RouteID, StopID,NumberofStops, ArrivalTime
Employee	EmployeeID, DepartmentID, FirstName,LastName, Position

Department	DepartmentID, DepartmentName, Location
Schedule	ScheduleID, BusID, RouteID, DayFlag, DepartureTime, ArrivalTime
Bus Stop	StopID, StopName, Location
Notification	NotificationID, Message, DateAndTime, UserID, AdminID, ReservationID
Holidays	HolidayID, HolidayDate, Description,NotificationID

6.1 IMPLEMENTATION OF ER DIAGRAM

Entities and Attributes:

Entity: User

Description: A table to describe the passengers in the Dutchess CountyBus Transportation System.

Attributes:

UserID – A unique identifier to identify each user. (Primary Key)FirstName – First name of the user.

LastName – Last name of the user. DOB – The date of birth of the user.Email – E-mail ID of the user.

PhoneNumber – The contact number of the user.

Username – The username of the user with which they want to login.Password – Password of the user.

Entity: Admin

Description: A table to describe the administrators who handle all the changes in the DCBTS.

Attributes:

AdminID – A unique identifier to identify each admin. (Primary Key)
Username – The username of the admin with which they want to login.Password – The password of the admin.
Email – E-mail ID of the admin.
PhoneNumber – The contact number of the admin. EmployeeID – The employee identifier of the admin.

Entity: Bus

Description: A table to describe all the properties of a bus.

Attributes:

BusID – A unique identifier to identify each bus. (Primary Key)

BusNumber – A designated bus number to identify the route of the bus.Capacity – The seating capacity of the bus.

LicensePlateNumber – The license plate number of the bus.BusType – The type of the bus. E.g.: Coach

Entity: Ticket Type

Description: A table to describe the type of each ticket.

Attributes:

TicketTypeID – A unique identifier to identify each ticket type.(Primary Key)

TypeName – Type name of the ticket. E.g.: Student ticketPrice – Price of the ticket.

ValidityPeriod – Validity period of the ticket.

Description – A description to explain which ticket applies to what age group.

UserID – (Foreign Key) to relate ticket type and user.

Entity: Payment

Description: A table to describe the payments for tickets.

Attributes:

PaymentID – A unique identifier for each payment. (Primary Key)Amount – Amount paid in the payment.

UserID – Identifier of the user who made the payment. (Foreign Key)

TicketTypeID – Identifier of the ticket type for which the payment has been made. (Foreign Key)

PaymentDate – The date on which the payment is made.

Entity: Reservation

Description: A table to describe all the seating reservations on the bus.

Attributes:

ReservationID – A unique identifier for each reservation. (Primary Key)

PaymentID – Payment identifier with which the seating reservation was made. (Foreign Key)

BusID – Identifier of the bus on which the reservation is made. (Foreign Key)

ReservationDate – The date for which the reservation is made.

NumberOfReservations – No. of reservations that have been made.

Entity: Bus Route

Description: A table to describe all the bus routes in the Dutchess County Bus Transportation System.

Attributes:

RouteID – A unique identifier to identify each route. (Primary Key)

StartLocation – Start location of the route.

EndLocation – End location of the route.

BusID – Identifier of the bus that will go on that route. (Foreign Key)

Entity: Route Stop Sequence

Description: A table to describe the route stop sequence from a source to a destination.

Attributes:

StopSequenceID – A unique identifier to identify a route stop sequence. (Primary Key)

RouteID – Route identifier to which the stop sequence is associated. (ForeignKey)

StopID – Identifier of every stop in the stop sequence. (Foreign Key)

NumberofStops – No. of stops in the route sequence.

ArrivalTime – The time at which the destination will arrive.

Entity: Employee

Description: A table to describe all the employees employed in the DCBTS.

Attributes:

EmployeeID – A unique identifier to identify each employee. (Primary Key) DepartmentID – Identifier of the department in which the employee is employed. (Foreign Key)

FirstName – First name of the employee.LastName – Last name of the employee.

Position – Designation of the employee.

Entity: Department

Description: A table to describe all the departments in the DCBTS.

Attributes:

DepartmentID – A unique identifier to identify each department. (PrimaryKey)

DepartmentName – Name of the department.Location – Location of the department.

Entity: Schedule

Description: A table to describe the bus schedule.

Attributes:

ScheduleID – An identifier to identify a schedule. (Primary Key)BusID – Identifier of the bus. (Foreign Key)

RouteID – Identifier of the routes. (Foreign Key)DayFlag – Flag to identify the day of week.

DepartureTime – Time when the bus departs.ArrivalTime – Time when the bus arrives.

Entity: Bus Stop

Description: A table to describe all the bus stops in DCBTS.

Attributes:

StopID – A unique identifier to identify each stop. (Primary Key)StopName – The name of the stop.

Location – The location of the stop.

Entity: Notification

Description: A table to describe all the notifications that address festivalsand other holidays.

Attributes:

NotificationID – A unique identifier to identify each notification. (PrimaryKey)

HolidayID – Identifier of the holiday to which the notification is associated. (Foreign Key)

ReservationID– Identifier of the reservation that will trigger a notification. (Foreign Key)

Message – Message content of the notification.

DateAndTime – Date and Time of when the notification will be displayed.

UserID – The identifier of the user who would see the notification. (ForeignKey)

AdminID – The identifier of the admin who would create the notification. (Foreign Key)

Entity: Holidays

Description: A table to describe the holidays during which the buses willnot be available.

Attributes:

HolidayID – A unique identifier to identify each holiday. (Primary Key)

HolidayDate – The date on which the holiday falls.

Description – Description of the holiday.

NotificationID – Identifier of the notification that will display the about this holiday. (Foreign Key)

- **Multivalued Attributes:**
  1. In the Ticket Type table, Price and ValidityPeriod can have multiple attributes.
  2. In the Route Stop Sequence table, the NumberofStops attribute is multi-valued as it indicates the count of stops on a route.

3. In the Reservation table, the NumberOfReservations attribute is multi-valued.

- **Composite Attributes:**

1. In the User and Employee table, we have User’s name and Employee’s name which will be a combination of FirstName and LastName.

- **Derived Attributes:**

1. In the Payment table, the Amount attribute can be derived from the Price in the Ticket Type.

- **Weak entity:**

1. Notification: The "Notification" entity is a weak entity because itrelies on either "Admin" or "User" for its existence.

- **Strong entity:**

1. Bus, Bus Route, Ticket Type, User, Bus Stop, Schedule, Holidays, Route stop sequence, Payment, Reservation, Employee, and Department are strong entities that do not depend on any other entitiesfor their existence or identity.

Participations:

Total Participation:

1. Each payment must be associated with a user, indicating total participation.
2. Each department should be associated with Employee, indicating total participation.
3. Each route stop sequence should be associated with bus stop, indicating total participation.
4. Each bus can be associated with many routes in different schedules same vice versa, which indicates Bus, Route, Schedule a total participation.

Partial Participation:

1. Users may or may not have associated records in other tables.
2. Not all administrators need to have associated employee records.
3. Not all holidays may have associated notification records.
4. Bus may or may not be associated with reservation.
5. Not all notifications need to be associated with User and Admin.
6. Not all reservations may be associated with notification records.

Cardinality Ratios:

1-N Cardinality Ratio:

- User: One user can make multiple payments (1-N cardinality).
- Notification: One admin or user can have multiple notifications (1-N cardinality).
- Payment: One user can make multiple payments (1-N cardinality).
- Reservation: One user can makemultiple reservations (1-Ncardinality).
- Employee: One department can have multiple employees (1-Ncardinality).
- BusStop: Each bus stop can be involved with multiple route stopsequence (1-N cardinality).

1-1 Cardinality Ratio:

- Reservation: Each reservation is associated with one notification (1-1 cardinality).
- Holidays: Each holiday is associated with one notification (1-1cardinality).

M-N Cardinality Ratio:

- Route stop sequence: Many routes can have many stops.
- Bus: Multiple buses can be associated with multiple routes.
- Payment: multiple payments lead to multiple reservations.
- Schedule: Different schedules linked to available buses.

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
- Romboids - 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:

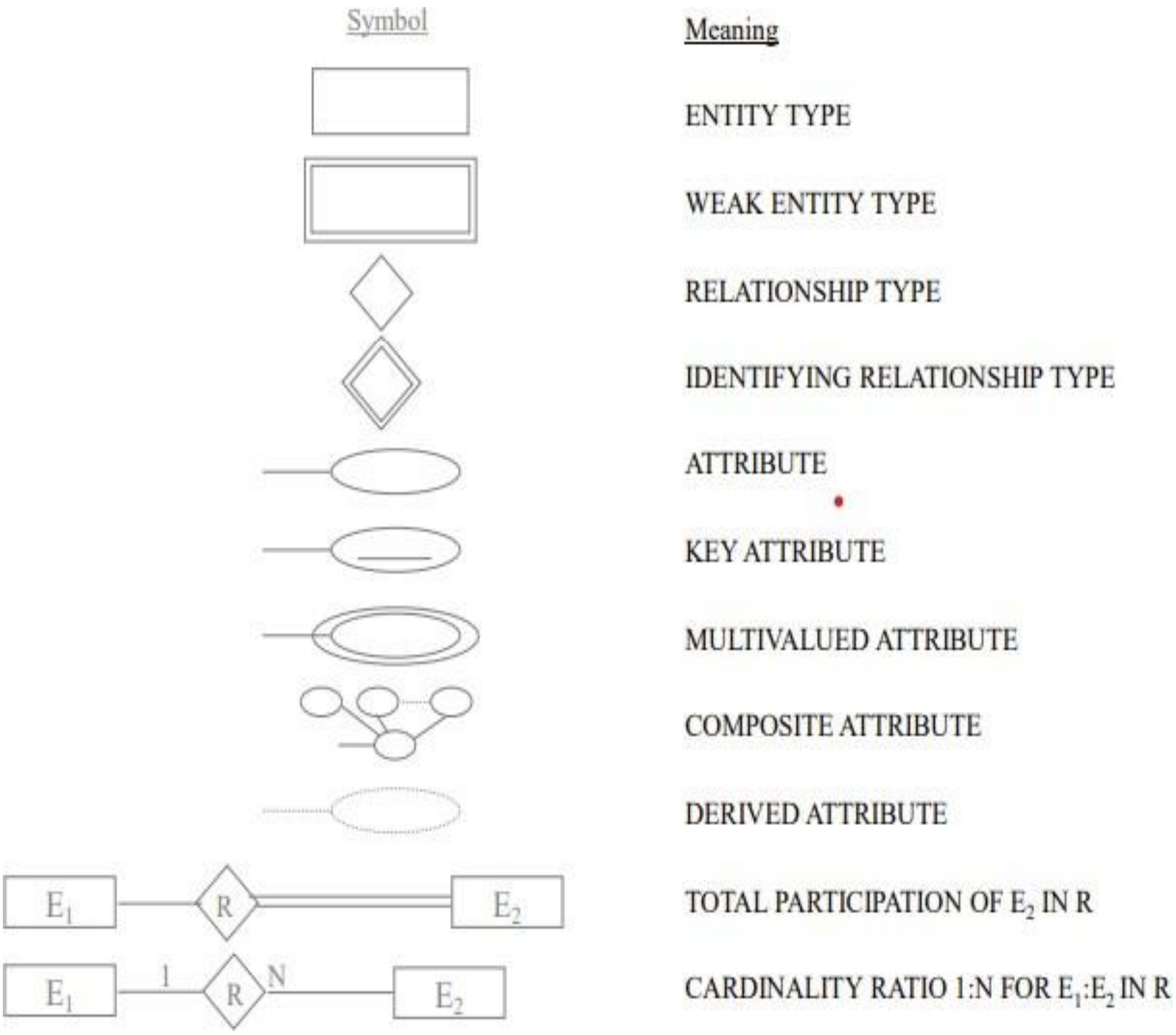


Figure 6. 1 Relationship of Entity Relationship Model

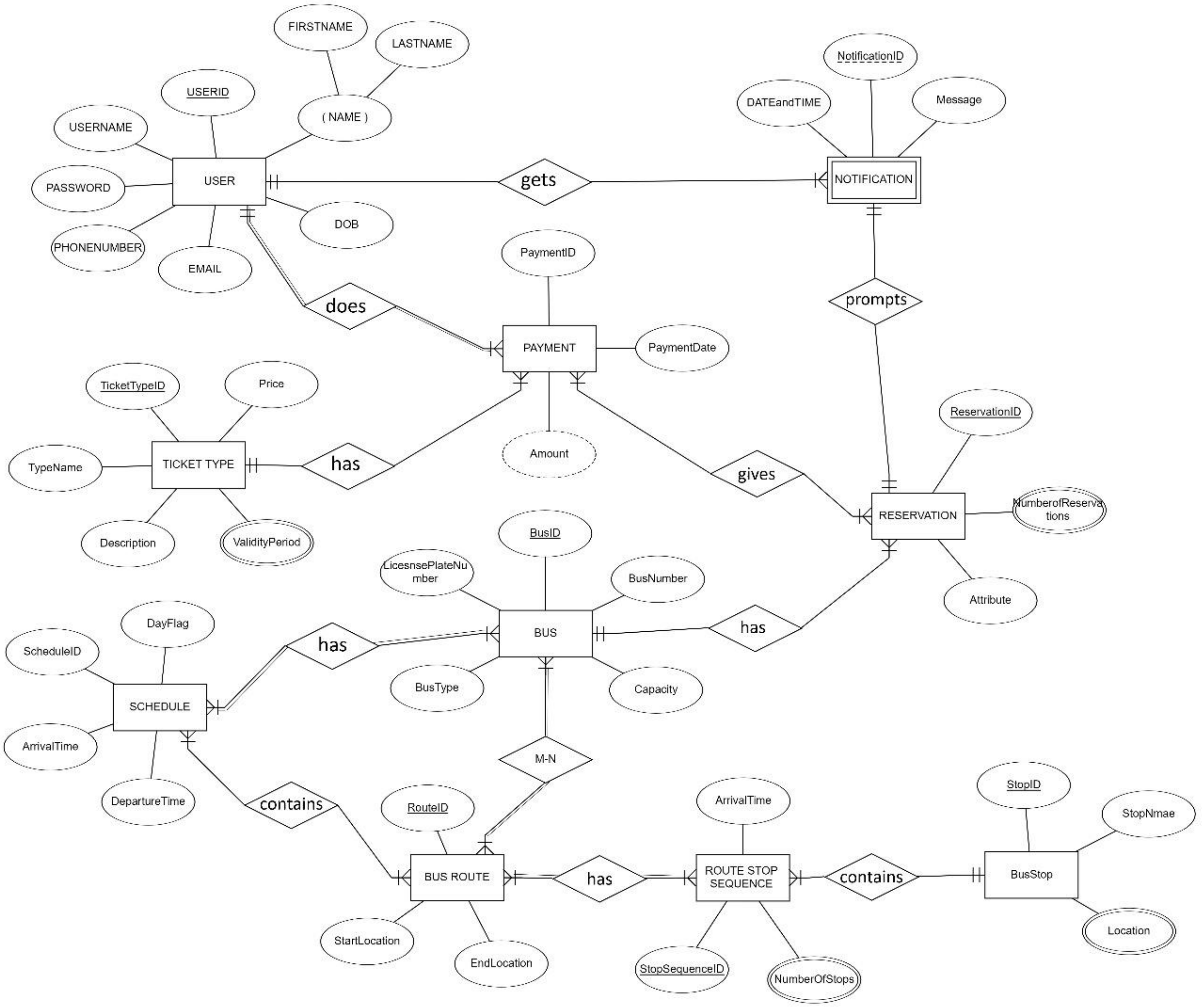


Figure 6. 2 External ER USER POV

As an external ER user, I can see that the system is designed to help me manage my bus reservations. I can use the system to view available bus routes and schedules, make reservations, and pay for my tickets. I can also use the system to track my reservations and receive notifications about changes to my schedule.

The system is easy to use and navigate. I can quickly find the information I need and make reservations with just a few clicks. I appreciate that the system provides me with real-time information about bus availability and schedules. This helps me to plan my trips and make sure that I am on time for my reservations.

I am also impressed with the system's security features. I can feel confident that my personal information is safe and secure when I make a reservation. Overall, I am very satisfied with the external ER user experience of this system.

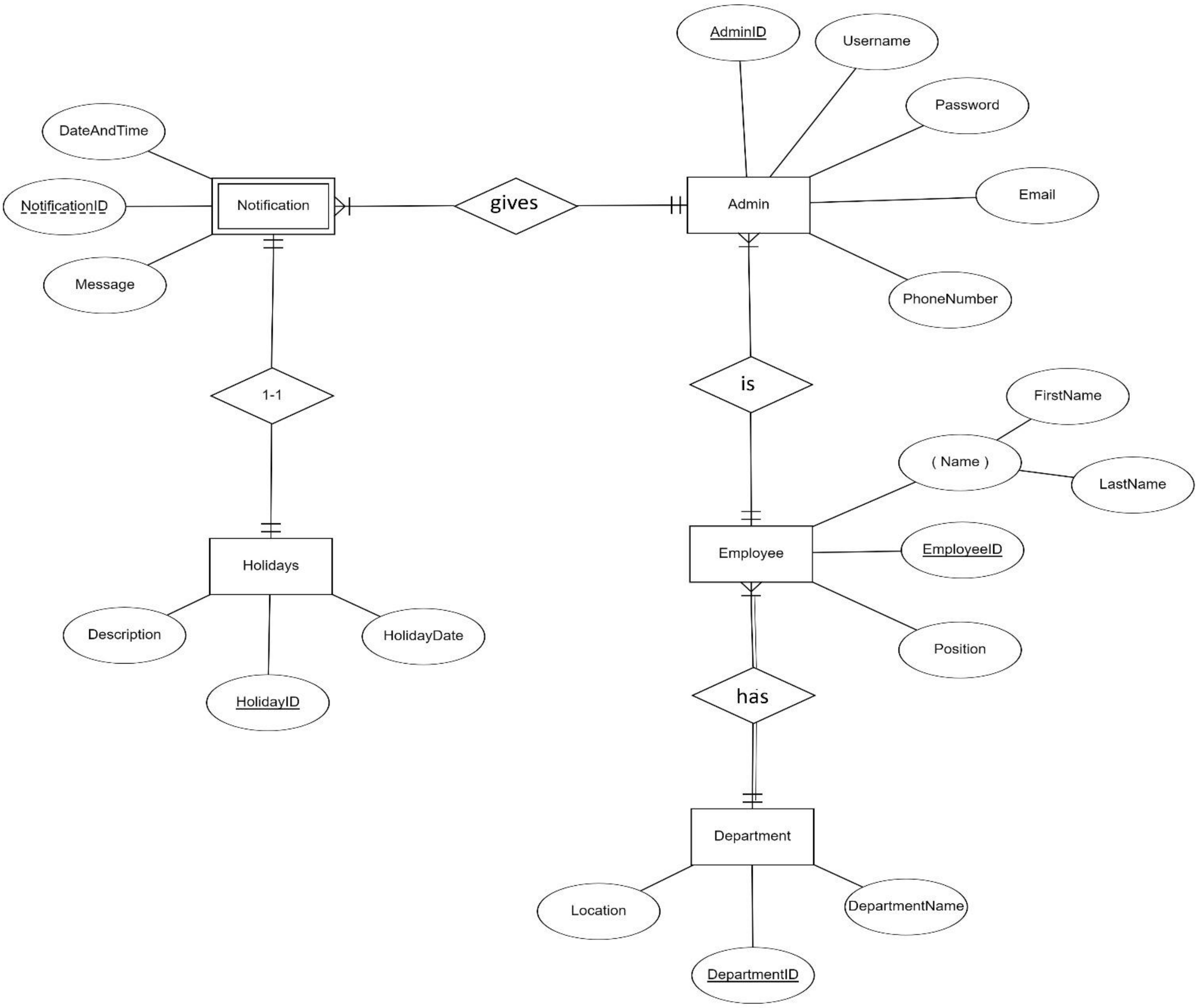


Figure 6. 3 External ER ADMIN POV

As an external ER admin, I am responsible for managing the system that allows employees to communicate with each other. This system is critical to our organization, as it allows employees to stay connected and collaborate on projects.

The system is easy to use and navigate. Employees can quickly find the information they need and communicate with each other with just a few clicks. I appreciate that the system provides me with real-time information about employee activity. This helps me to identify and address any potential issues early on.

I am also impressed with the system's security features. I can feel confident that our employees' communications are safe and secure. Overall, I am very satisfied with the external ER admin experience of this system.

The system is well-designed and meets the needs of our organization. I am confident that it will continue to be a valuable tool for our employees in the future.

ENTITY RELATIONSHIP DIAGRAM

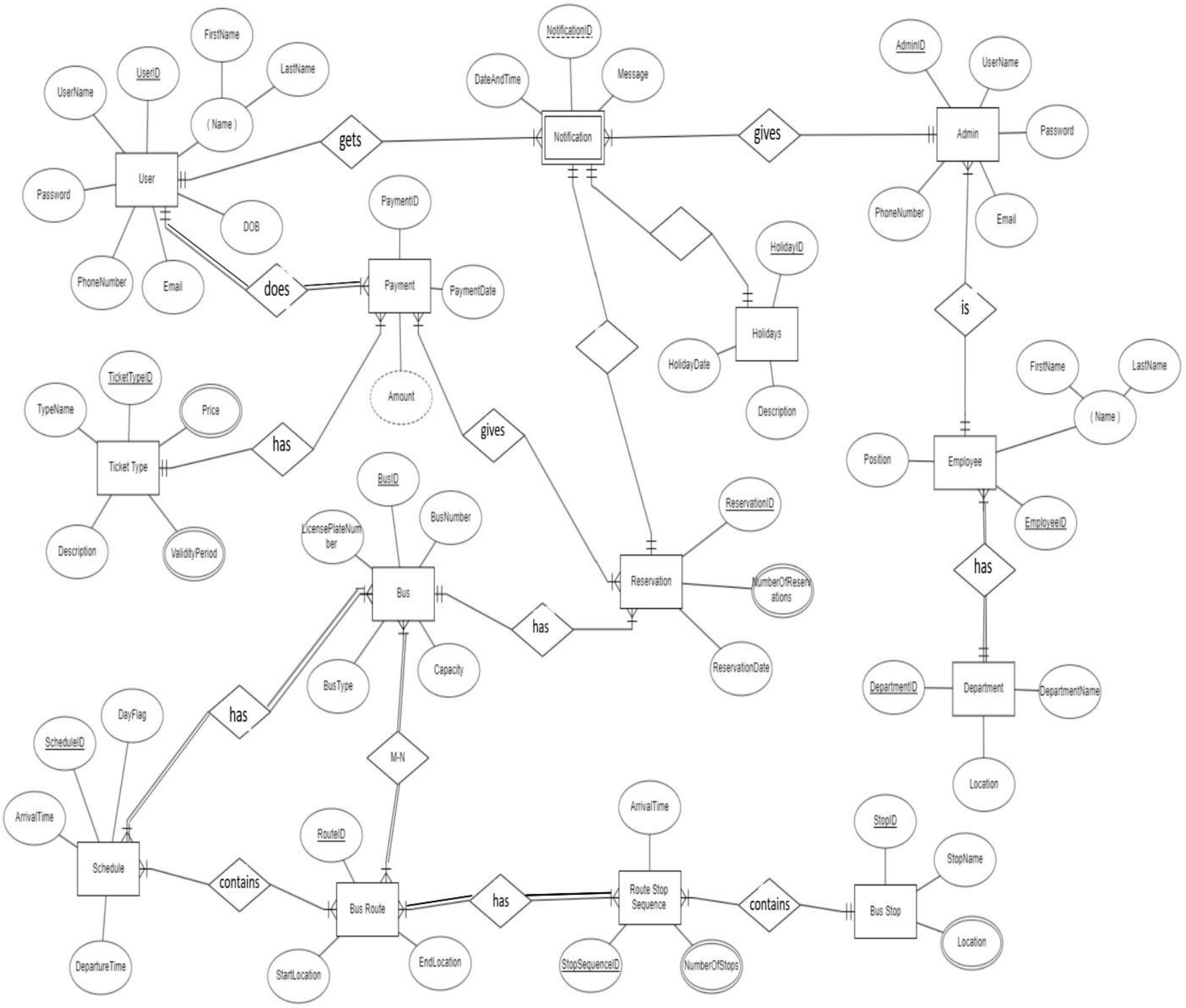


Figure 6. 4 Entity Relationship Diagram

7.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.

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.

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.

When the application became complex, the traditional ER model was not enough to draw a sophisticated diagram. Therefore, the ER model was developed further. It is known as the Enhanced ER diagram. There are threeconcepts added to the existing ER model in the Enhanced ER diagram (EER). Those are generalization, specialization, and aggregation. In generalization, the lower-level entities can be combined to produce a higher- level entity. Specialization is the opposite of generalization. In specialization, the high-level entities can be divided into lower-level entities.Aggregation is a process when the relation between two entities is treated asa single entity.

Additionally, it includes the concepts of a subclass and superclass (Is-a). Furthermore, it introduces the concept of a union type or category, which represents a collection of objects that is the union of objects of different entity types. The EER model also includes EER diagrams which are conceptual models that accurately represent the requirements of complex databases.

7.1 IMPLEMENTATION OF EER

This Enhanced Entity Relationship (EER) diagram offers a comprehensive model for a complex system comprising essential entities, including payment, employee, admin, ticket type, time, date, last name, department, name, and Sunday Bus schedules. Each entity plays a distinctive role within the database structure. Payment, for instance, is intricately connected to a specific ticket type, employee, and date, ensuring precise payment records. Meanwhile, employees and admins are associated with departments and identified by their First name and Last name, forming the backbone of personnel management.

Ticket types are uniquely defined by their names and prices, preventing any duplications, while time and date entities offer precise temporal and date- based representations. Names act as versatile identifiers across various domains, connecting entities within the system. Additionally, Sunday schedules are distinctly linked to a department, date, and time for efficient planning.

The relationships between these entities are systematically outlined, ensuringdata integrity and retrieval efficiency. For example, strict associations like payments being tied to one ticket type, employee, and date eliminate any chances of duplicate payments. This design not only safeguards data but alsostreamlines data retrieval, making it easier to extract meaningful informationfrom the database.

In conclusion, this well-designed EER diagram offers an efficient model forthe system's organization. It ensures data integrity, enhances data retrieval performance, and provides clarity, making it a valuable tool for managing and organizing complex data structures within the described system.

ENHANCED ENTITY RELATIONSHIP DIAGRAM

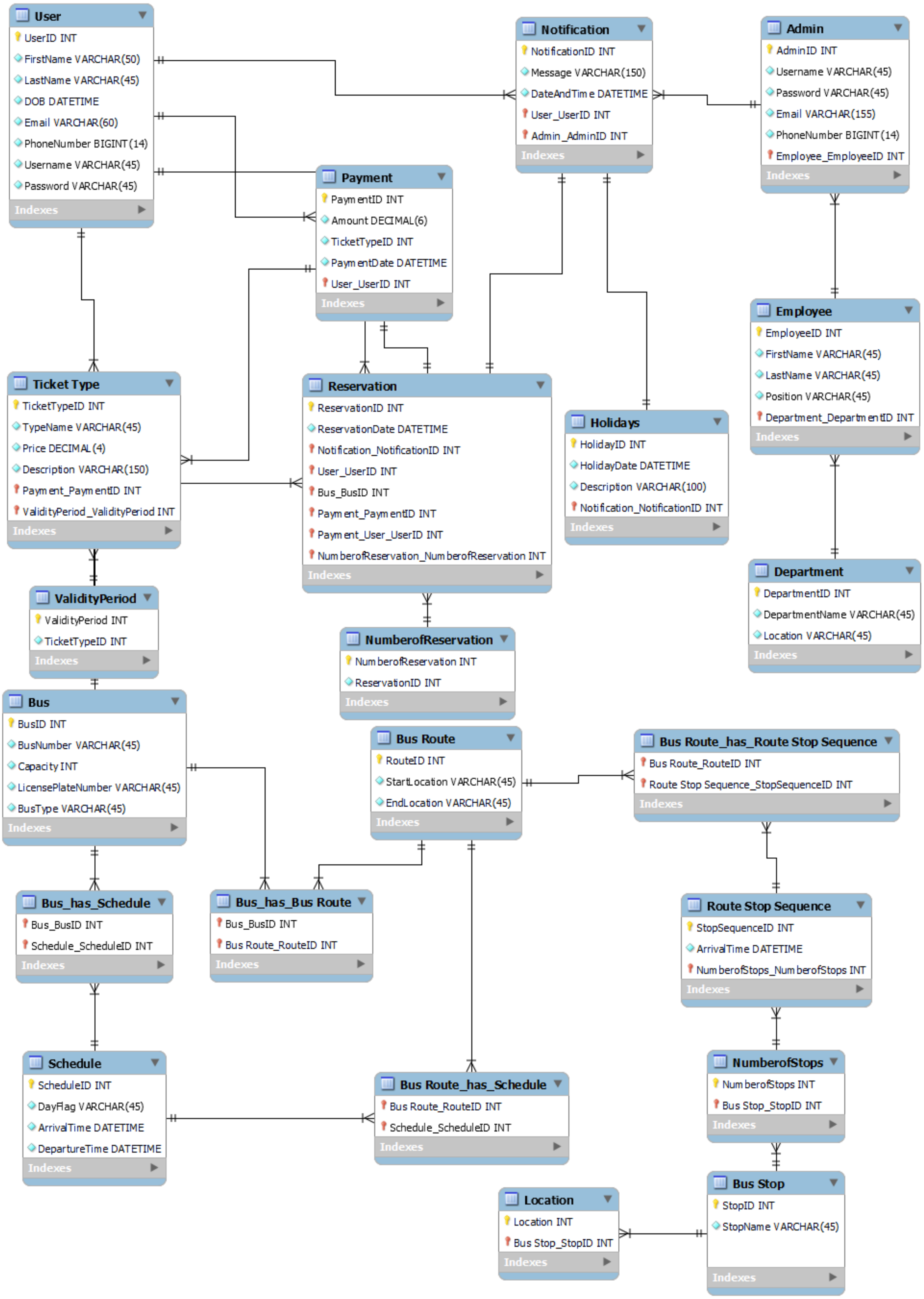


Figure 7. 1 Enhanced Entity Relationship

8. 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, meets all the primary and foreign key constraints.

8.1 DESCRIPTION:

Table Name	Description
User	**Usage:** User table defines a user in the DCBTS. **Data attributes:** 1. UserID INT 2. FirstName VARCHAR(50) 3. LastName VARCHAR(45) 4. DOB DATETIME 5. Email VARCHAR(60) 6. PhoneNumber BIGINT(14) 7. Username VARCHAR(45) 8. Password VARCHAR(45) **Constraints:** 1. PRIMARY KEY – UserID **Relationships:** 1. A User can make many Payments. 2. A User can receive many Notifications.
Bus	**Usage:** Bus table defines a bus in the DCBTS. **Data attributes:** 1. BusID INT 2. BusNumber VARCHAR(45) 3. Capacity INT 4. LicensePlateNumber VARCHAR(45) 5. BusType VARCHAR(45) **Constraints:** 1. PRIMARY KEY- BusID **Relationships:** 1. A User can make many Payments. 2. A User can receive many Notifications.

Department	**Usage:** Department table defines the departments in the DCBTS. **Data Attributes:** 1. DepartmentID INT 2. DepartmentName VARCHAR(45) 3. Location VARCHAR(45) **Contraints:** 1. PRIMARY KEY- DepartmentID **Relationships:** 1. A Department has many Employees.
Employee	**Usage:** Employee table defines an employee in the DCBTS. **Data Attributes:** 1. EmployeeID INT 2. FirstName VARCHAR(45) 3. LastName VARCHAR(45) 4. Position VARCHAR(45) 5. Department_DepartmentID INT **Contraints:** 1. PRIMARY KEY-EmployeeID, Department_DepartmentID 2. FOREIGN KEY-Department(DepartmentID) **Relationships:** 1. An Employee works in one Department. 2. An Employee can be linked to one Admin account.
Admin	**Usage:** Admin table defines an admin in the DCBTS. **Data Attributes:** 1. AdminID INT 2. Username VARCHAR(45) 3. Password VARCHAR(45) 4. Email VARCHAR(155) 5. PhoneNumber BIGINT(14) 6. Employee_EmployeeID INT **Constraints:** 1. PRIMARY KEY- AdminID, Employee_EmployeeID 2. FOREIGN KEY - Employee (EmployeeID) **Relationships:** 1. An Admin is linked to one Employee.

	2. An Admin can send many Notifications.
Payment	**Usage:** Payment table defines a payment made in the DCBTS. **Data Attributes:** 1. PaymentID INT 2. Amount DECIMAL(6) 3. PaymentDate DATETIME 4. User_UserID` INT 5. Ticket Type_TicketTypeID INT **Constraints:** 1. PRIMARY KEY-PaymentID, User_UserID, TicketType_TicketTypeID 2. FOREIGN KEY – User(UserID), TicketType(TicketTypeID) **Relationships:** 1. A Payment is made by one User. 2. A Payment is for one TicketType. 3. A Payment can be linked to many Reservations.
Schedule	**Usage:** Schedule table defines a schedule in the DCBTS. **Data Attributes:** 1. ScheduleID INT NOT NULL 2. DayFlag VARCHAR(45) 3. ArrivalTime DATETIME 4. DepartureTime DATETIME **Constraints:** 1. PRIMARY KEY- ScheduleID **Relationships:** 1. A Schedule can be assigned to many Buses. 2. A Schedule can be assigned to many Routes.
BusRoute	**Usage:** BusRoute defines a bus route in the DCBTS. **Data Attributes:** 1. RouteID INT 2. StartLocation VARCHAR(45) 3. EndLocation VARCHAR(45) **Constraints:** 1. PRIMARY KEY-RouteID **Relationships:** 1. A Route has many RouteStopSequences. 2. A Route can be assigned many Schedules.

	3. A Route can have many Buses assigned to it.
BusStop	**Usage:** BusStop table defines a bus stop in the DCBTS. **Data Attributes:** 1. StopID INT 2. StopName VARCHAR(45) 3. Location VARCHAR(45) **Constraints:** 1. PRIMARY KEY-StopID **Relationships:** 1. One (BusStop) to Many (RouteStopSequences). 2. One bus stop can be assigned to many route stop sequences.
RouteStopSequence	**Usage:** Route Stop Sequence table defines the sequence of the stops in a route. **Data Attributes:** 1. StopSequenceID INT 2. NumberofStops INT 3. ArrivalTime DATETIME 4. Bus Stop_StopID INT **Constraints:** 1. PRIMARY KEY-StopSequenceID, Bus Stop_StopID 2. FOREIGN KEY- BusStop(StopID) **Relationships:** 1. Many (RouteStopSequences) to One (BusRoute). 2. One bus route has many route stop sequences.
TicketType	**Usage:** TicketType table defines the types of tickets in DCBTS. **Data Attributes:** 1. TicketTypeID INT 2. TypeName VARCHAR(45) 3. Price DECIMAL(4) 4. ValidityPeriod INT 5. Description VARCHAR(150) **Constraints:** 1. PRIMARY KEY-TicketTypeID **Relationships:**

	1. One (TicketType) to Many (Payments). 2. One ticket type can be used for many payments.
Notification	**Usage:** Notification table defines a notification of important events in the DCBTS. **Data Attributes:** 1. NotificationID INT 2. Message VARCHAR(150) 3. DateAndTime DATETIME 4. User_UserID INT 5. Admin_AdminID INT **Constraints:** 1. PRIMARY KEY-NotificationID, User_UserID, Admin_AdminID 2. FOREIGN KEY- User(UserID), Admin(AdminID) **Relationships:** 1. A Notification is sent by one Admin. 2. A Notification is sent to one User. 3. A Notification can be linked to many Holidays. 4. A Notification can be linked to many Reservations.
Holidays	**Usage:** Holidays table defines the holidays in DCBTS. **Data Attributes:** 1. HolidayID INT 2. HolidayDate DATETIME 3. Description VARCHAR(100) 4. Notification_NotificationID INT **Constraints:** 1. PRIMARY KEY-HolidayID, Notification_NotificationID 2. FOREIGN KEY-Notification(NotificationID) **Relationships:** 1. Many (Holidays) to One (Notification). 2. One notification can be linked to many holidays.
Reservation	**Usage:** Reservation table defines a reservation made by the user in DCBTS.

	Data Attributes: 1. ReservationID INT 2. ReservationDate DATETIME 3. NumberOfReservations INT 4. Notification_NotificationID INT 5. Bus_BusID INT **Constraints:** 1. PRIMARY KEY-ReservationID, Notification_NotificationID, Bus_BusID 2. FOREIGN KEY-Notification(NotificationID), Bus(BusID) **Relationships:** 1. Many (Reservations) to One (Notification). 2. One notification can be linked to many reservations.
BusRoute_has_RouteStopSequence	**Usage:** Links one BusRoute to one RouteStopSequence. **Data Attributes:** 1. Bus Route_RouteID INT 2. RouteStopSequence_StopSequenceID INT **Constraints:** 1. PRIMARY KEY-BusRoute_RouteID, RouteStopSequence_StopSequenceID 2. FOREIGN KEY-BusRoute(RouteID),RouteStopSequence(StopSequenceID) **Realtionships:** 1. One bus route can be linked to many route stop sequences. 2. One route stop sequence can be linked to many bus routes.
Payment_has_Reservation	**Usage:** Links one Payment to one Reservation. **Data Attributes:** 1. Payment_PaymentID INT 2. Payment_User_UserID INT 3. Payment_Ticket Type_TicketTypeID INT 4. Reservation_ReservationID INT 5. Reservation_Notification_NotificationID INT 6. Reservation_Bus_BusID INT **Constraints:** 1. PRIMARY KEY-Payment_PaymentID,

	Payment_User_UserID, Payment_Ticket Type_TicketTypeID, Reservation_ReservationID, Reservation_Notification_NotificationID, Reservation_Bus_BusID 2. FOREIGN KEY- Payment(PaymentID, User_UserID,Ticket Type_TicketTypeID), Reservation(ReservationID,Notification_NotificationID,Bus_BusID) **Realtionships:** 1. One (Payment) to One (Reservation). 2. Links one payment to one reservation.
Bus_has_Schedule	**Usage:** Links one Bus to one Schedule. **Data Attributes:** 1. Bus_BusID INT 2. Schedule_ScheduleID INT **Constraints:** 1. PRIMARY KEY-Bus_BusID, Schedule_ScheduleID 2. FOREIGN KEY- Bus(BusID), Schedule(ScheduleID) **Realtionships:** 1. One bus can be assigned many schedules. 2. One schedule can be assigned to many buses.
Bus_has_BusRoute	**Usage:** Links one Bus to one BusRoute. **Data Attributes:** 1. Bus_BusID INT 2. Bus Route_RouteID INT **Constraints:** 1. PRIMARY KEY-Bus_BusID, Bus Route_RouteID 2. FOREIGN KEY -Bus(BusID), BusRoute(RouteID) **Realtionships:** 1. One bus can be assigned many routes 2. One route can have many buses assigned
BusRoute_has_Schedule	**Usage:** Links one BusRoute to one Schedule. **Data Attributes:** 1. Bus Route_RouteID INT

	2. Schedule_ScheduleID INT **Constraints:** 1. PRIMARY KEY-Bus Route_RouteID, Schedule_ScheduleID 2. FOREIGN KEY- Bus Route(RouteID), Schedule(ScheduleID) **Realtionships:** 1. One route can have many schedules. 2. One schedule can be assigned to many routes.

8.2 CREATE STATEMENTS:

1. Create Dutchess_county_bus_transportation_DBMS_project Database:

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

2. Create User Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`User` (
  `UserID` INT NOT NULL AUTO_INCREMENT,
  `FirstName` VARCHAR(50) NOT NULL,
  `LastName` VARCHAR(45) NOT NULL,
  `DOB` DATETIME NOT NULL,
  `Email` VARCHAR(60) NOT NULL,
  `PhoneNumber` BIGINT(14) NOT NULL,
  `Username` VARCHAR(45) NOT NULL,
  `Password` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`UserID`),
  UNIQUE INDEX `Username_UNIQUE` (`Username` ASC) VISIBLE,
  UNIQUE INDEX `UserID_UNIQUE` (`UserID` ASC) VISIBLE)
ENGINE = InnoDB;
```

3. Create Bus Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Bus` (
  `BusID` INT NOT NULL AUTO_INCREMENT,
  `BusNumber` VARCHAR(45) NOT NULL,
  `Capacity` INT NOT NULL,
  `LicensePlateNumber` VARCHAR(45) NOT NULL,
  `BusType` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`BusID`),
  UNIQUE INDEX `BusID_UNIQUE` (`BusID` ASC) VISIBLE,
  UNIQUE INDEX `BusNumber_UNIQUE` (`BusNumber` ASC) VISIBLE,
  UNIQUE INDEX `LicensePlateNumber_UNIQUE` (`LicensePlateNumber` ASC)
VISIBLE)
ENGINE = InnoDB;
```

4. Create BusRoute Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Bus Route` (
  `RouteID` INT NOT NULL AUTO_INCREMENT,
  `StartLocation` VARCHAR(45) NOT NULL,
  `EndLocation` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`RouteID`))
ENGINE = InnoDB;
```

5. Create TicketType Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Ticket Type` (
  `TicketTypeID` INT NOT NULL,
  `TypeName` VARCHAR(45) NOT NULL,
  `Price` DECIMAL(4) NOT NULL,
  `ValidityPeriod` INT NULL,
  `Description` VARCHAR(150) NOT NULL,
  PRIMARY KEY (`TicketTypeID`),
  UNIQUE INDEX `TicketTypeID_UNIQUE` (`TicketTypeID` ASC) VISIBLE,
  UNIQUE INDEX `TypeName_UNIQUE` (`TypeName` ASC) VISIBLE)
ENGINE = InnoDB;
```

6. Create Department Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Department` (
  `DepartmentID` INT NOT NULL AUTO_INCREMENT,
  `DepartmentName` VARCHAR(45) NOT NULL,
  `Location` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`DepartmentID`),
  UNIQUE INDEX `DepartmentID_UNIQUE` (`DepartmentID` ASC) VISIBLE)
ENGINE = InnoDB;
```

7. Create Employee Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Employee` (
  `EmployeeID` INT NOT NULL AUTO_INCREMENT,
  `FirstName` VARCHAR(45) NOT NULL,
  `LastName` VARCHAR(45) NOT NULL,
  `Position` VARCHAR(45) NOT NULL,
  `Department_DepartmentID` INT NOT NULL,
  PRIMARY KEY (`EmployeeID`, `Department_DepartmentID`),
  UNIQUE INDEX `EmployeeID_UNIQUE` (`EmployeeID` ASC) VISIBLE,
  INDEX `fk_Employee_Department1_idx` (`Department_DepartmentID` ASC) VISIBLE,
  CONSTRAINT `fk_Employee_Department1`
    FOREIGN KEY (`Department_DepartmentID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Department`
(`DepartmentID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

8. Create Admin Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Admin` (
  `AdminID` INT NOT NULL AUTO_INCREMENT,
  `Username` VARCHAR(45) NOT NULL,
  `Password` VARCHAR(45) NOT NULL,
  `Email` VARCHAR(155) NOT NULL,
  `PhoneNumber` BIGINT(14) NOT NULL,
  `Employee_EmployeeID` INT NOT NULL,
  PRIMARY KEY (`AdminID`, `Employee_EmployeeID`),
  UNIQUE INDEX `AdminID_UNIQUE` (`AdminID` ASC) VISIBLE,
  UNIQUE INDEX `Email_UNIQUE` (`Email` ASC) VISIBLE,
  UNIQUE INDEX `PhoneNumber_UNIQUE` (`PhoneNumber` ASC) VISIBLE,
  INDEX `fk_Admin_Employee1_idx` (`Employee_EmployeeID` ASC) VISIBLE,
  CONSTRAINT `fk_Admin_Employee1`
    FOREIGN KEY (`Employee_EmployeeID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Employee`
(`EmployeeID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

9. Create BusStop Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Bus Stop` (
  `StopID` INT NOT NULL AUTO_INCREMENT,
  `StopName` VARCHAR(45) NOT NULL,
  `Location` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`StopID`),
  UNIQUE INDEX `StopID_UNIQUE` (`StopID` ASC) VISIBLE)
ENGINE = InnoDB;
```

10. Create Notification Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Notification` (
  `NotificationID` INT NOT NULL AUTO_INCREMENT,
  `Message` VARCHAR(150) NOT NULL,
  `DateAndTime` DATETIME NOT NULL,
  `User_UserID` INT NOT NULL,
  `Admin_AdminID` INT NOT NULL,
  PRIMARY KEY (`NotificationID`, `User_UserID`, `Admin_AdminID`),
  UNIQUE INDEX `NotificationID_UNIQUE` (`NotificationID` ASC) VISIBLE,
  INDEX `fk_Notification_User1_idx` (`User_UserID` ASC) VISIBLE,
  INDEX `fk_Notification_Admin1_idx` (`Admin_AdminID` ASC) VISIBLE,
  CONSTRAINT `fk_Notification_User1`
    FOREIGN KEY (`User_UserID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`User` (`UserID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Notification_Admin1`
    FOREIGN KEY (`Admin_AdminID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Admin`
(`AdminID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

11. Create Holidays Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Holidays` (
  `HolidayID` INT NOT NULL AUTO_INCREMENT,
  `HolidayDate` DATETIME NOT NULL,
  `Description` VARCHAR(100) NOT NULL,
  `Notification_NotificationID` INT NOT NULL,
  PRIMARY KEY (`HolidayID`, `Notification_NotificationID`),
  INDEX `fk_Holidays_Notification1_idx` (`Notification_NotificationID` ASC) VISIBLE,
  CONSTRAINT `fk_Holidays_Notification1`
    FOREIGN KEY (`Notification_NotificationID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Notification`
(`NotificationID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

12. Create RouteStopSequence Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Route Stop Sequence` (
  `StopSequenceID` INT NOT NULL AUTO_INCREMENT,
  `NumberofStops` INT NOT NULL,
  `ArrivalTime` DATETIME NOT NULL,
  `Bus Stop_StopID` INT NOT NULL,
  PRIMARY KEY (`StopSequenceID`, `Bus Stop_StopID`),
  UNIQUE INDEX `StopSequenceID_UNIQUE` (`StopSequenceID` ASC) VISIBLE,
  INDEX `fk_Route Stop Sequence_Bus Stop1_idx` (`Bus Stop_StopID` ASC) VISIBLE,
  CONSTRAINT `fk_Route Stop Sequence_Bus Stop1`
    FOREIGN KEY (`Bus Stop_StopID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Bus Stop`
(`StopID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

13. Create Payment Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Payment` (
  `PaymentID` INT NOT NULL,
  `Amount` DECIMAL(6) NOT NULL,
  `PaymentDate` DATETIME NOT NULL,
  `User_UserID` INT NOT NULL,
  `Ticket Type_TicketTypeID` INT NOT NULL,
  PRIMARY KEY (`PaymentID`, `User_UserID`, `Ticket Type_TicketTypeID`),
  INDEX `fk_Payment_User1_idx` (`User_UserID` ASC) VISIBLE,
  INDEX `fk_Payment_Ticket Type1_idx` (`Ticket Type_TicketTypeID` ASC) VISIBLE,
  CONSTRAINT `fk_Payment_User1`
    FOREIGN KEY (`User_UserID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`User` (`UserID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Payment_Ticket Type1`
    FOREIGN KEY (`Ticket Type_TicketTypeID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Ticket Type`
(`TicketTypeID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

14. Create Reservation Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Reservation` (
  `ReservationID` INT NOT NULL AUTO_INCREMENT,
  `ReservationDate` DATETIME NOT NULL,
  `NumberOfReservations` INT NOT NULL,
  `Notification_NotificationID` INT NOT NULL,
  `Bus_BusID` INT NOT NULL,
  PRIMARY KEY (`ReservationID`, `Notification_NotificationID`, `Bus_BusID`),
  INDEX `fk_Reservation_Notification1_idx` (`Notification_NotificationID` ASC) VISIBLE,
  INDEX `fk_Reservation_Bus1_idx` (`Bus_BusID` ASC) VISIBLE,
  CONSTRAINT `fk_Reservation_Notification1`
    FOREIGN KEY (`Notification_NotificationID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Notification`
(`NotificationID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Reservation_Bus1`
    FOREIGN KEY (`Bus_BusID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Bus` (`BusID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

15. Create Bus_has_Bus Route Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Bus_has_Bus Route` (
  `Bus_BusID` INT NOT NULL,
  `Bus Route_RouteID` INT NOT NULL,
  PRIMARY KEY (`Bus_BusID`, `Bus Route_RouteID`),
  INDEX `fk_Bus_has_Bus Route_Bus Route1_idx` (`Bus Route_RouteID` ASC) VISIBLE,
  INDEX `fk_Bus_has_Bus Route_Bus1_idx` (`Bus_BusID` ASC) VISIBLE,
  CONSTRAINT `fk_Bus_has_Bus Route_Bus1`
    FOREIGN KEY (`Bus_BusID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Bus` (`BusID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Bus_has_Bus Route_Bus Route1`
    FOREIGN KEY (`Bus Route_RouteID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Bus Route`
(`RouteID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

16. Create Bus Route_has_RouteStopSequence Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Bus Route_has_Route Stop
Sequence` (
  `Bus Route_RouteID` INT NOT NULL,
  `Route Stop Sequence_StopSequenceID` INT NOT NULL,
  PRIMARY KEY (`Bus Route_RouteID`, `Route Stop Sequence_StopSequenceID`),
  INDEX `fk_Bus Route_has_Route Stop Sequence_Route Stop Sequence1_idx` (`Route Stop
Sequence_StopSequenceID` ASC) VISIBLE,
  INDEX `fk_Bus Route_has_Route Stop Sequence_Bus Route1_idx` (`Bus Route_RouteID`
ASC) VISIBLE,
  CONSTRAINT `fk_Bus Route_has_Route Stop Sequence_Bus Route1`
    FOREIGN KEY (`Bus Route_RouteID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Bus Route`
(`RouteID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Bus Route_has_Route Stop Sequence_Route Stop Sequence1`
    FOREIGN KEY (`Route Stop Sequence_StopSequenceID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Route Stop
Sequence` (`StopSequenceID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

17. Create Schedule Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Schedule` (
  `ScheduleID` INT NOT NULL AUTO_INCREMENT,
  `DayFlag` VARCHAR(45) NOT NULL,
  `ArrivalTime` DATETIME NOT NULL,
  `DepartureTime` DATETIME NOT NULL,
  PRIMARY KEY (`ScheduleID`),
  UNIQUE INDEX `ScheduleID_UNIQUE` (`ScheduleID` ASC) VISIBLE)
ENGINE = InnoDB;
```

18. Create Bus_has_Schedule Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Bus_has_Schedule` (
  `Bus_BusID` INT NOT NULL,
  `Schedule_ScheduleID` INT NOT NULL,
  PRIMARY KEY (`Bus_BusID`, `Schedule_ScheduleID`),
  INDEX `fk_Bus_has_Schedule_Schedule1_idx` (`Schedule_ScheduleID` ASC) VISIBLE,
  INDEX `fk_Bus_has_Schedule_Bus1_idx` (`Bus_BusID` ASC) VISIBLE,
  CONSTRAINT `fk_Bus_has_Schedule_Bus1`
    FOREIGN KEY (`Bus_BusID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Bus` (`BusID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Bus_has_Schedule_Schedule1`
    FOREIGN KEY (`Schedule_ScheduleID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Schedule`
(`ScheduleID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

19. Create Bus Route_has_Schedule Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Bus Route_has_Schedule` (
  `Bus Route_RouteID` INT NOT NULL,
  `Schedule_ScheduleID` INT NOT NULL,
  PRIMARY KEY (`Bus Route_RouteID`, `Schedule_ScheduleID`),
  INDEX `fk_Bus Route_has_Schedule_Schedule1_idx` (`Schedule_ScheduleID` ASC)
VISIBLE,
  INDEX `fk_Bus Route_has_Schedule_Bus Route1_idx` (`Bus Route_RouteID` ASC)
VISIBLE,
  CONSTRAINT `fk_Bus Route_has_Schedule_Bus Route1`
    FOREIGN KEY (`Bus Route_RouteID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Bus Route`
(`RouteID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Bus Route_has_Schedule_Schedule1`
    FOREIGN KEY (`Schedule_ScheduleID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Schedule`
(`ScheduleID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

20. Create Payment_has_Reservation Table:

```
CREATE TABLE IF NOT EXISTS
`Dutchess_county_bus_transportation_DBMS_project`.`Payment_has_Reservation` (
  `Payment_PaymentID` INT NOT NULL,
  `Payment_User_UserID` INT NOT NULL,
  `Payment_Ticket Type_TicketTypeID` INT NOT NULL,
  `Reservation_ReservationID` INT NOT NULL,
  `Reservation_Notification_NotificationID` INT NOT NULL,
  `Reservation_Bus_BusID` INT NOT NULL,
  PRIMARY KEY (`Payment_PaymentID`, `Payment_User_UserID`, `Payment_Ticket
Type_TicketTypeID`, `Reservation_ReservationID`,
`Reservation_Notification_NotificationID`, `Reservation_Bus_BusID`),
  INDEX `fk_Payment_has_Reservation_Reservation1_idx` (`Reservation_ReservationID`
ASC, `Reservation_Notification_NotificationID` ASC, `Reservation_Bus_BusID` ASC)
VISIBLE,
  INDEX `fk_Payment_has_Reservation_Payment1_idx` (`Payment_PaymentID` ASC,
`Payment_User_UserID` ASC, `Payment_Ticket Type_TicketTypeID` ASC) VISIBLE,
  CONSTRAINT `fk_Payment_has_Reservation_Payment1`
    FOREIGN KEY (`Payment_PaymentID` , `Payment_User_UserID` , `Payment_Ticket
Type_TicketTypeID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Payment`
(`PaymentID` , `User_UserID` , `Ticket Type_TicketTypeID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Payment_has_Reservation_Reservation1`
    FOREIGN KEY (`Reservation_ReservationID` ,
`Reservation_Notification_NotificationID` , `Reservation_Bus_BusID`)
    REFERENCES `Dutchess_county_bus_transportation_DBMS_project`.`Reservation`
(`ReservationID` , `Notification_NotificationID` , `Bus_BusID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

9. LOADING DATA AND PERFORMANCE ENHANCEMENTS

9.1 HANDLING FOREIGN KEY CONSTRAINTS:

Case 1:

a) Insert parent Department records first

```
INSERT INTO Department (DepartmentName, Location)
VALUES
  ('Operations', '123 Main St'),
  ('Maintenance', '456 Park Rd');
```

b) Then insert child Employee records

```
INSERT INTO Employee (FirstName, Department_DepartmentID)
VALUES
  ('John', 1),
  ('Jane', 2);
```

c) Disable FK checks

```
SET foreign_key_checks = 0;
```

d) Insert Bus records with invalid Department_ID

```
INSERT INTO Employee (FirstName, Department_DepartmentID) VALUES ('Bob',
3);
```

e) Enable FK checks again

```
SET foreign_key_checks = 1;
```

Description:

- The Department table is the parent table of the Employee table via the Department_DepartmentID foreign key.
- The Department records are inserted first to ensure the foreign key references exist before inserting child records.
- The Employee records that reference DepartmentIDs are then inserted.
- Foreign key checks are disabled to allow inserting an Employee with an invalid foreign key reference.
- Foreign key checks are re-enabled after insertion is complete to restore validation.

Case 2:

```
Error Code: 1452. Cannot add or update a child row: a foreign key constraint
fails
(`dutchess_county_bus_transportation_dbms_project`.`bus_has_schedule`,
CONSTRAINT `fk_Bus_has_Schedule_Bus1` FOREIGN KEY
(`Bus_BusID`) REFERENCES `bus` (`BusID`))        0.047 sec
```

We used two ways after deletion :

```
SET SQL_SAFE_UPDATES = 0;
DELETE FROM Bus;
SET SQL_SAFE_UPDATES = 1;
DELETE FROM Bus
WHERE BusID > 0;
```

```
a)  ALTER TABLE Bus AUTO_INCREMENT = 10;

b)  INSERT INTO Bus (BusID, BusNumber, Capacity, LicensePlateNumber, BusType)
    VALUES
      (1, 'A', 50, 'ABC126','Coach'),
      (2, 'B', 60, 'GHI789', 'School Bus'),
      (3, 'C', 40, 'JKL012', 'Shuttle'),
      (4, 'D', 45, 'MNO345', 'Shuttle'),
      (5, 'E', 35, 'PQR678', 'Mini Bus'),
      (6, 'F', 30, 'STU901', 'Mini Bus'),
      (7, 'G', 35, 'STU905', 'Bus'),
      (8, 'H', 39, 'STU910', 'Mini Bus');
```

Description:

- To overcome this issue, we deleted the existing records and did this in two ways.
- By using the Alter table bus and by checking the schedule table with the bus table and updating the records manually.

9.2 IMPORTING DATA:

We took the data from the DCPT and entered it manually in the respective .csv file and imported that data safely using the following.

1. Import data from Bus.csv into Bus table

   ```
   LOAD DATA LOCAL INFILE 'Bus.csv'
   INTO TABLE Bus
   FIELDS TERMINATED BY ','
   ENCLOSED BY '"'
   LINES TERMINATED BY '\n'
   IGNORE 1 ROWS;
   ```

2. Import data from BusRoute.csv into BusRoute table

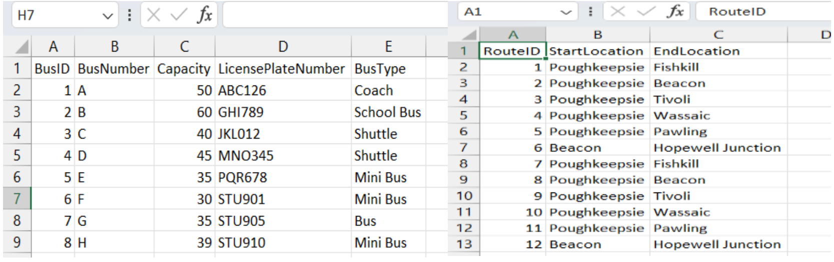
   ```
   LOAD DATA LOCAL INFILE 'BusRoute.csv'
   INTO TABLE BusRoute
   FIELDS TERMINATED BY ','
   ENCLOSED BY '"'
   LINES TERMINATED BY '\n'
   IGNORE 1 ROWS;
   ```

3. Import data from TicketType.csv into TicketType table

   ```
   LOAD DATA LOCAL INFILE 'TicketType.csv'
   INTO TABLE TicketType
   FIELDS TERMINATED BY ','
   ENCLOSED BY '"'
   LINES TERMINATED BY '\n'
   IGNORE 1 ROWS;
   ```

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.



	A	B	C	D	E
1	BusID	BusNumber	Capacity	LicensePlateNumber	BusType
2	1	A	50	ABC126	Coach
3	2	B	60	GHI789	School Bus
4	3	C	40	JKL012	Shuttle
5	4	D	45	MNO345	Shuttle
6	5	E	35	PQR678	Mini Bus
7	6	F	30	STU901	Mini Bus
8	7	G	35	STU905	Bus
9	8	H	39	STU910	Mini Bus

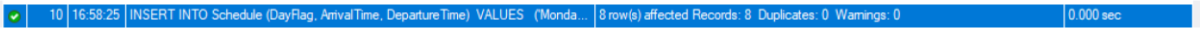
	A	B	C	D
1	RouteID	StartLocation	EndLocation	
2	1	Poughkeepsie	Fishkill	
3	2	Poughkeepsie	Beacon	
4	3	Poughkeepsie	Tivoli	
5	4	Poughkeepsie	Wassaic	
6	5	Poughkeepsie	Pawling	
7	6	Beacon	Hopewell Junction	
8	7	Poughkeepsie	Fishkill	
9	8	Poughkeepsie	Beacon	
10	9	Poughkeepsie	Tivoli	
11	10	Poughkeepsie	Wassaic	
12	11	Poughkeepsie	Pawling	
13	12	Beacon	Hopewell Junction	

Figure 9.3 Usage of CSV files to load data

9.3 INSERTION OPTIMIZATION:

- Used bulk INSERT statements instead of separate INSERTs:

```
INSERT INTO Schedule (DayFlag, ArrivalTime, DepartureTime)
VALUES
  ('Monday', '06:00:00', '06:05:00'),
  ('Monday', '06:10:00', '06:15:00'),
  ('Tuesday', '07:00:00', '07:05:00'),
  ('Wednesday', '08:30:00', '08:35:00'),
  ('Thursday', '15:45:00', '15:50:00'),
  ('Friday', '13:10:00', '13:15:00'),
  ('Saturday', '11:15:00', '11:20:00'),
    ('Sunday', '09:30:00', '09:35:00');
```



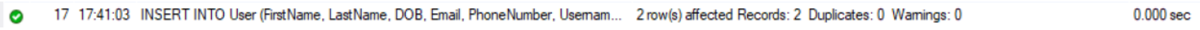
```
✓  10  16:58:25  INSERT INTO Schedule (DayFlag, ArrivalTime, DepartureTime)  VALUES  ('Monda...  8 row(s) affected Records: 8  Duplicates: 0  Warnings: 0  0.000 sec
```

- Disabled indexes and foreign key checks before insertion:

```
SET foreign_key_checks = 0;
ALTER TABLE Bus DISABLE KEYS;

-- Insert statements

ALTER TABLE Bus ENABLE KEYS;
  SET foreign_key_checks = 1;
```

- Used multiple VALUES lists in one INSERT statement:

```
INSERT INTO User (FirstName, LastName, DOB, Email, PhoneNumber,
Username, Password)
VALUES
  ('John', 'Doe', '1990-01-01', 'john@example.com', '123-456-7890', 'johndoe',
'password123'),
    ('Jane', 'Smith', '1995-05-15', 'jane@example.com', '987-654-3210', 'janesmith',
  'password456');
```



```
✓  17  17:41:03  INSERT INTO User (FirstName, LastName, DOB, Email, PhoneNumber, Usernam...  2 row(s) affected Records: 2  Duplicates: 0  Warnings: 0  0.000 sec
```

- Imported data from CSV using LOAD DATA instead of INSERTs:

```
LOAD DATA LOCAL INFILE 'Bus.csv'
INTO TABLE Bus
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

This way we optimized insertion by reducing context switches, utilizing bulk inserts, and minimizing lagging.

9.4 INSERT STATEMENTS:

1. Insert schedules:

```
INSERT INTO Schedule (DayFlag, ArrivalTime, DepartureTime)
VALUES
  ('Monday', '06:00:00', '06:05:00'),
  ('Monday', '06:10:00', '06:15:00'),
  ('Tuesday', '07:00:00', '07:05:00'),
  ('Wednesday', '08:30:00', '08:35:00'),
  ('Thursday', '15:45:00', '15:50:00'),
  ('Friday', '13:10:00', '13:15:00'),
  ('Saturday', '11:15:00', '11:20:00'),
  ('Sunday', '09:30:00', '09:35:00');
```

2. Insert bus routes:

```
INSERT INTO BusRoute (StartLocation, EndLocation)
VALUES
  ('Poughkeepsie', 'Fishkill'),
  ('Poughkeepsie', 'Beacon'),
  ('Poughkeepsie', 'Tivoli'),
  ('Poughkeepsie', 'Wassaic'),
  ('Poughkeepsie', 'Pawling'),
  ('Beacon', 'Hopewell Junction');
```

3. Insert bus stops:

```
INSERT INTO BusStop (StopName, Location)
VALUES
  ('Poughkeepsie Station', '1 Station Plaza, Poughkeepsie, NY'),
  ('Beacon Station', '223 Fishkill Ave, Beacon NY'),
  ('Hopewell Junction', '123 Main St, Hopewell Junction, NY'),
  ('Tivoli', '456 Broadway, Tivoli, NY'),
  ('Fishkill', '789 Main St, Fishkill, NY'),
  ('Wassaic', '234 Rail Rd, Wassaic, NY');
```

4. Insert bus stop arrival times:

```
INSERT INTO RouteStopSequence (NumberofStops, ArrivalTime, BusStop_StopID)
VALUES
  (1, '06:00:00', 1),
  (2, '06:15:00', 2),
  (3, '06:30:00', 3),
  (4, '06:45:00', 4),
  (5, '07:00:00', 5),
  (6, '07:15:00', 6);
```

5. Assign buses to schedules:

```
INSERT INTO Bus_has_Schedule (Bus_BusID, Schedule_ScheduleID)
VALUES
  (1, 1),
  (2, 2),
  (3, 3),
  (4, 4),
  (5, 5),
  (6, 6),
  (7, 7),
  (8, 8);
```

6. Assign buses to routes:

```
INSERT INTO Bus_has_BusRoute (Bus_BusID, BusRoute_RouteID)
VALUES
  (1, 1),
  (2, 2),
  (17, 3),
  (18, 4),
  (19, 5),
  (20, 6);
```

7. Insert ticket types:

```
INSERT INTO TicketType (TicketTypeID, TypeName, Price, ValidityPeriod, Description)
VALUES
  (1, 'Day Pass', 5.00, 1, 'Valid for one day'),
  (2, '7-day Pass', 20.00, 7, 'Valid for 7 consecutive days'),
  (3, '31-day Pass', 70.00, 31, 'Valid for 31 consecutive days');
```

8. Insert data into User table:

```
INSERT INTO User (FirstName, LastName, DOB, Email, PhoneNumber, Username,
Password)
VALUES
  ('John', 'Doe', '1990-01-01', 'john@example.com', '123-456-7890', 'johndoe', 'password123'),
  ('Jane', 'Smith', '1995-05-15', 'jane@example.com', '987-654-3210', 'janesmith',
'password456');
```

9. Insert data into the Bus table:

```
INSERT INTO Bus (BusID, BusNumber, Capacity, LicensePlateNumber, BusType)
VALUES
(1, 'A', 50, 'ABC126','Coach'),
(2, 'B', 60, 'GHI789', 'School Bus'),
(3, 'C', 40, 'JKL012', 'Shuttle'),
(4, 'D', 45, 'MNO345', 'Shuttle'),
(5, 'E', 35, 'PQR678', 'Mini Bus'),
(6, 'F', 30, 'STU901', 'Mini Bus'),
(7, 'G', 35, 'STU905', 'Bus'),
```

10.Insert data into the Department table:

```
INSERT INTO Department (DepartmentName, Location)
VALUES
  ('Operations', '123 Main St, Poughkeepsie, NY'),
  ('Maintenance', '456 Park Rd, Beacon, NY');
```

11. Insert data into Employee table:

```
INSERT INTO Employee (FirstName, LastName, Position, Department_DepartmentID)
VALUES
  ('Sarah', 'Jones', 'Driver', 1),
  ('Mark', 'Lee', 'Mechanic', 2);
```

12. Insert data into Admin table:

```
INSERT INTO Admin (Username, Password, Email, PhoneNumber,
Employee_EmployeeID)
VALUES
  ('sarahj','password123', 'sarah@example.com','123-555-1234',1),
  ('markl','password456', 'mark@example.com','987-555-4321',2);
```

13. Insert data into Payment table:

```
INSERT INTO Payment (PaymentID, Amount, PaymentDate, User_UserID,
TicketType_TicketTypeID)
VALUES
  (1, 5.00, '2023-01-01', 1, 1),
  (2, 20.00, '2023-01-10', 2, 2);
```

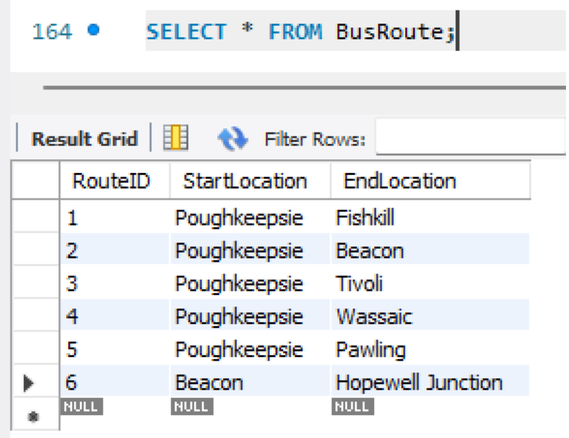
9.5 NORMALIZATION CHECK

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

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

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

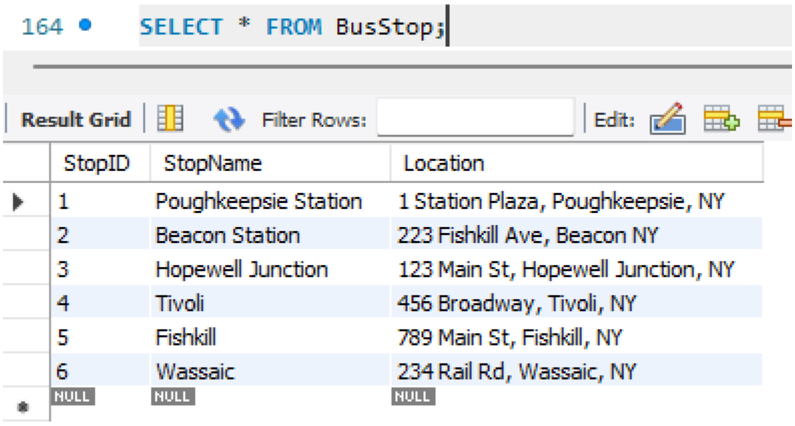
- For example, in the BusRoute table, all the values are unique and each cell contains only a single value.



```
SELECT * FROM BusRoute;
```

RouteID	StartLocation	EndLocation
1	Poughkeepsie	Fishkill
2	Poughkeepsie	Beacon
3	Poughkeepsie	Tivoli
4	Poughkeepsie	Wassaic
5	Poughkeepsie	Pawling
6	Beacon	Hopewell Junction
NULL	NULL	NULL

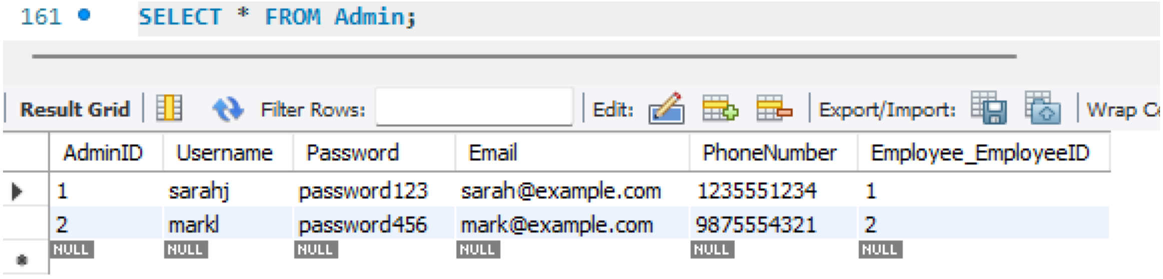
- For example, in the BusStop table, all the values are unique and each cell contains only a single value.



```
SELECT * FROM BusStop;
```

StopID	StopName	Location
1	Poughkeepsie Station	1 Station Plaza, Poughkeepsie, NY
2	Beacon Station	223 Fishkill Ave, Beacon NY
3	Hopewell Junction	123 Main St, Hopewell Junction, NY
4	Tivoli	456 Broadway, Tivoli, NY
5	Fishkill	789 Main St, Fishkill, NY
6	Wassaic	234 Rail Rd, Wassaic, NY
NULL	NULL	NULL

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



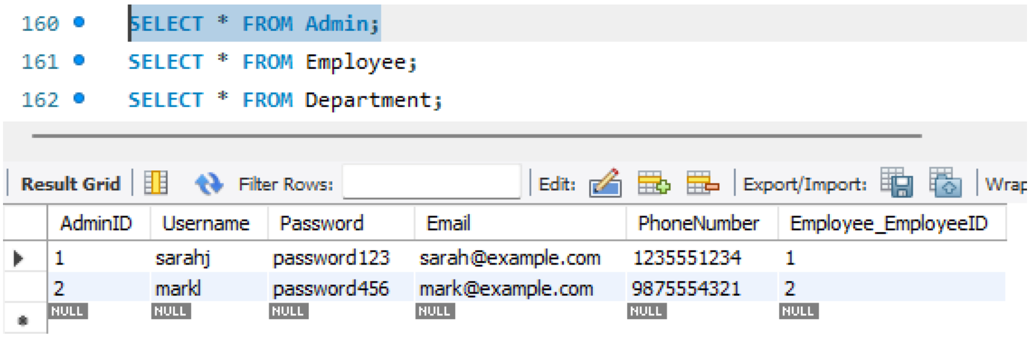
```sql
SELECT * FROM Admin;
```

AdminID	Username	Password	Email	PhoneNumber	Employee_EmployeeID
1	sarahj	password123	sarah@example.com	1235551234	1
2	markl	password456	mark@example.com	9875554321	2
NULL	NULL	NULL	NULL	NULL	NULL

- **Second Normal Form(2NF):**

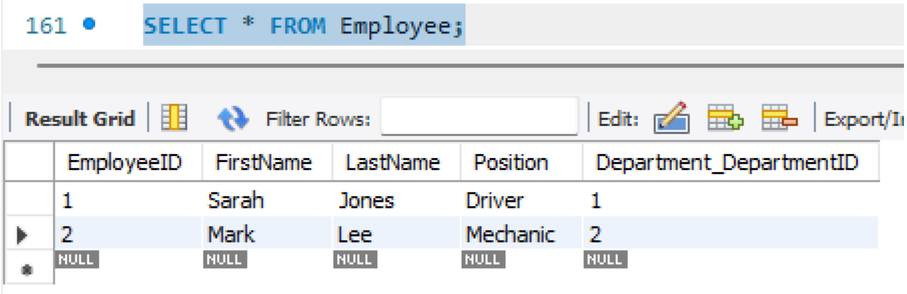
The second normal form states that it should be in 1NF and it does not have any non-prime attribute that is functionally dependent on any proper subset of any candidate key of the relation.

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



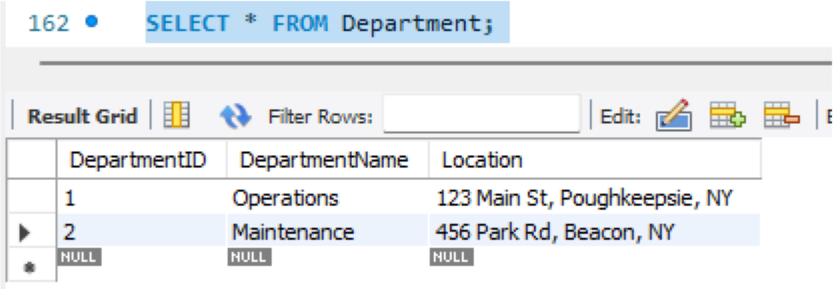
```sql
SELECT * FROM Admin;
SELECT * FROM Employee;
SELECT * FROM Department;
```

AdminID	Username	Password	Email	PhoneNumber	Employee_EmployeeID
1	sarahj	password123	sarah@example.com	1235551234	1
2	markl	password456	mark@example.com	9875554321	2
NULL	NULL	NULL	NULL	NULL	NULL



```sql
SELECT * FROM Employee;
```

EmployeeID	FirstName	LastName	Position	Department_DepartmentID
1	Sarah	Jones	Driver	1
2	Mark	Lee	Mechanic	2
NULL	NULL	NULL	NULL	NULL



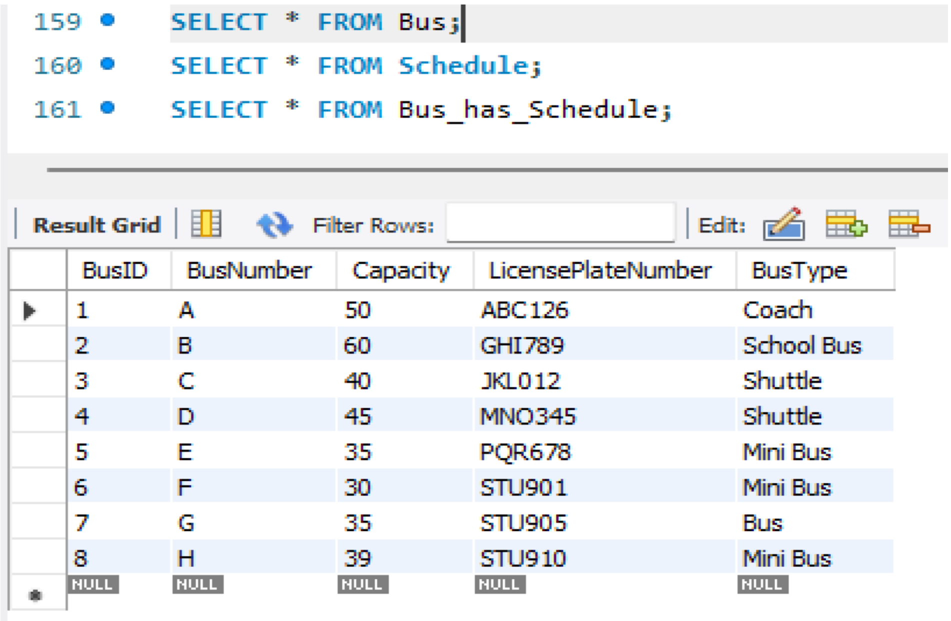
```sql
SELECT * FROM Department;
```

DepartmentID	DepartmentName	Location
1	Operations	123 Main St, Poughkeepsie, NY
2	Maintenance	456 Park Rd, Beacon, NY
NULL	NULL	NULL

- **Third Normal Form(3NF):**

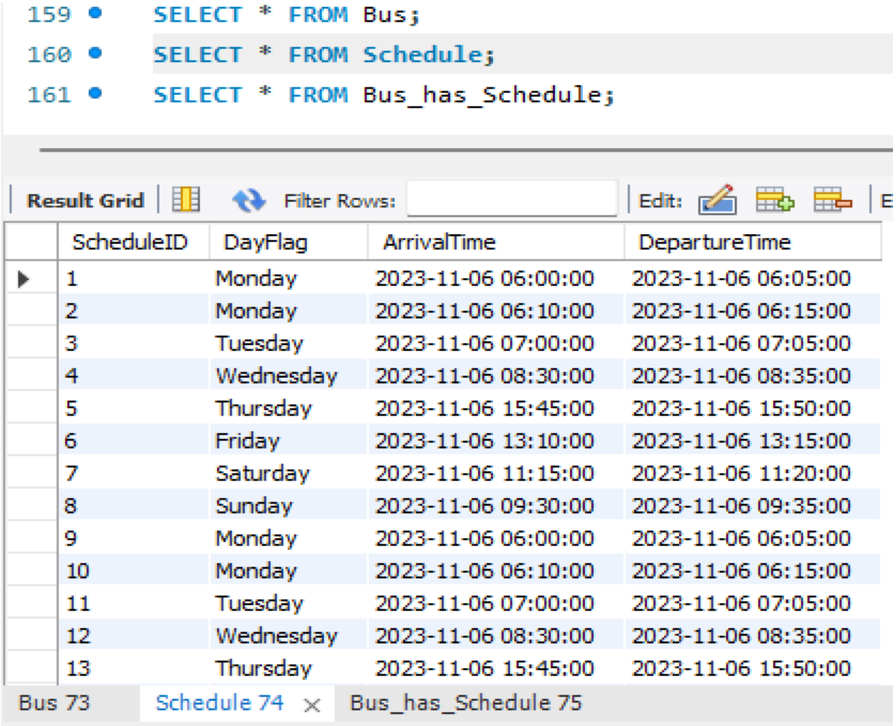
The third normal form states that it should be in 2NF and has no transitive functional dependencies.

- For example, in the tables Bus and Schedule there exists a transitive dependency between each bus and each schedule. Hence, we created a new table called Bus_has_Schedule to remove the transitive dependency, which has BusID as the primary key.



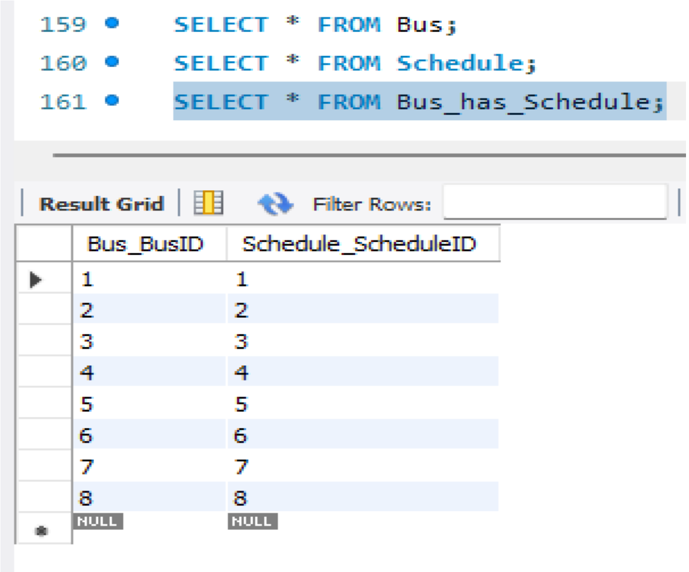
```
SELECT * FROM Bus;
SELECT * FROM Schedule;
SELECT * FROM Bus_has_Schedule;
```

BusID	BusNumber	Capacity	LicensePlateNumber	BusType
1	A	50	ABC126	Coach
2	B	60	GHI789	School Bus
3	C	40	JKL012	Shuttle
4	D	45	MNO345	Shuttle
5	E	35	PQR678	Mini Bus
6	F	30	STU901	Mini Bus
7	G	35	STU905	Bus
8	H	39	STU910	Mini Bus
NULL	NULL	NULL	NULL	NULL



```
SELECT * FROM Bus;
SELECT * FROM Schedule;
SELECT * FROM Bus_has_Schedule;
```

ScheduleID	DayFlag	ArrivalTime	DepartureTime
1	Monday	2023-11-06 06:00:00	2023-11-06 06:05:00
2	Monday	2023-11-06 06:10:00	2023-11-06 06:15:00
3	Tuesday	2023-11-06 07:00:00	2023-11-06 07:05:00
4	Wednesday	2023-11-06 08:30:00	2023-11-06 08:35:00
5	Thursday	2023-11-06 15:45:00	2023-11-06 15:50:00
6	Friday	2023-11-06 13:10:00	2023-11-06 13:15:00
7	Saturday	2023-11-06 11:15:00	2023-11-06 11:20:00
8	Sunday	2023-11-06 09:30:00	2023-11-06 09:35:00
9	Monday	2023-11-06 06:00:00	2023-11-06 06:05:00
10	Monday	2023-11-06 06:10:00	2023-11-06 06:15:00
11	Tuesday	2023-11-06 07:00:00	2023-11-06 07:05:00
12	Wednesday	2023-11-06 08:30:00	2023-11-06 08:35:00
13	Thursday	2023-11-06 15:45:00	2023-11-06 15:50:00



```
SELECT * FROM Bus;
SELECT * FROM Schedule;
SELECT * FROM Bus_has_Schedule;
```

Bus_BusID	Schedule_ScheduleID
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
NULL	NULL

10. APPLICATION DEVELOPMENT

10.1 GRAPHICAL USER EXPERIENCE DESIGN

10.1.1 GUE ADMIN POV

The flowchart depicts a process from the admin's point of view for an online bus reservation system.

It starts with Admin Login where the admin enters their username and password. If invalid, it shows an error.

On successful login, the admin lands on the Landing Page which is likely the admin dashboard.

From the dashboard, the admin can access:

Bus Route Table - To view, add, edit bus routes

Notifications - To view notifications sent to users

Holidays - To mark holidays that affect schedules

Authentication - For account security settings

The flow indicates the admin needs to be authenticated before accessing the admin-only sections like Bus Route Table, Notifications, and Holidays.

Without authentication, the admin remains on the public Landing Page. With valid authentication, they can access private admin sections.

In summary, the flowchart outlines the admin login process and key pages of the bus reservation back-end system accessible to the authenticated admin user.

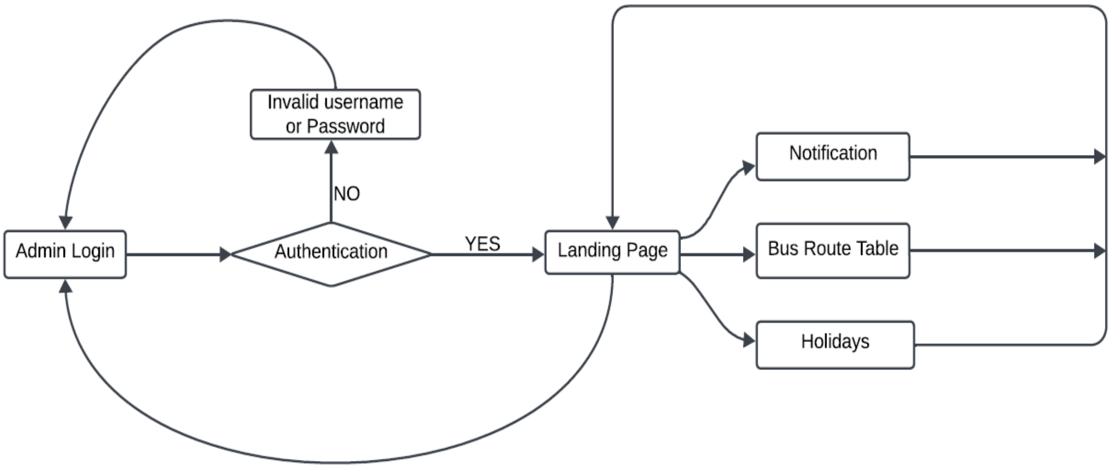


Figure 10. 1.1 GUI Admin POV

10.1.2 GUE USER POV

The flowchart outlines the user journey on the bus reservation website from login to searching and booking.

It begins with the Login Landing Page where the user enters their username and password.

If the user is new, they can sign up for an account.

If they enter an invalid username or password, an error is displayed.

Upon successful login, users reach the main Landing Page with options like:

Authentication - To manage account security
Search - To search for bus routes
Payment - To book and pay for a bus reservation
Notifications - To view booking confirmations and alerts
Reservation ID - To access booking reference codes
From the Landing Page, the user can perform a Search. If invalid, an error shows.

With a valid search, it flows to the Search Validation page showing available bus routes and timings.

The user can select a route and proceed to Payment to complete the bus reservation.

Upon successful payment, the user receives a Reservation ID notification.

At any point, the user can Exit the website from the Landing Page or Search Results.

In summary, the flow outlines the end-to-end user experience - from login, to search, booking payment, and notifications for bus ticket reservations on the website.

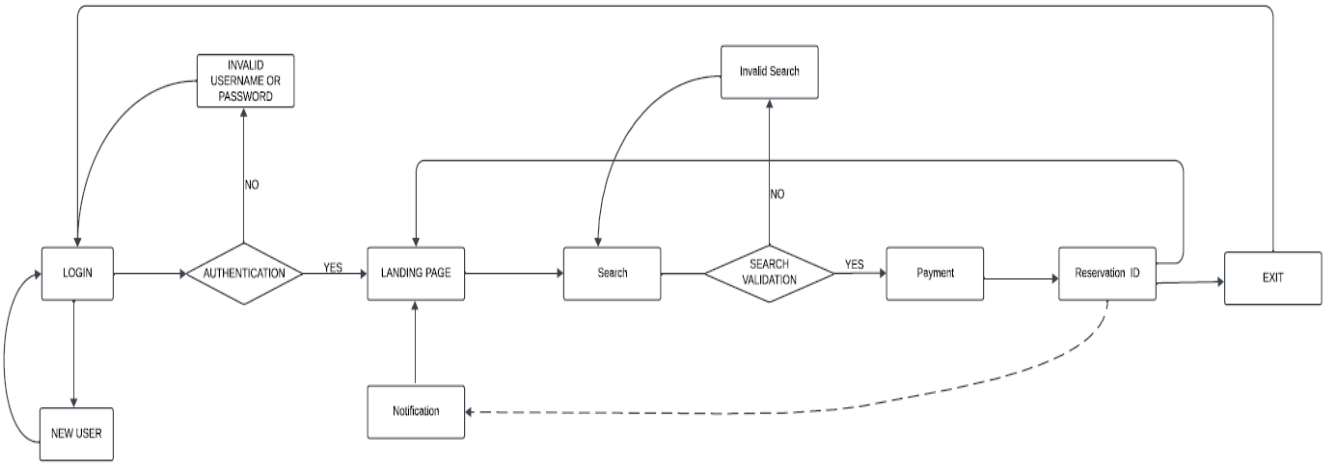


Figure 10.1.2 GUI User POV

10.2 VIEWS IMPLEMENTATION

The provided code is creating a series of database views in a database named "dutchess_county_bus_transportation_dbms_project." These views are used to simplify data retrieval and provide specific information from the underlying tables. Here's a brief explanation of each view created in the code:

1. `v_user_info`: Retrieves information about users, including their UserId, FirstName, LastName, Date of Birth (DOB), Email, Phone Number, and Username from the "User" table.

2. `v_bus_routes`: Extracts data related to bus routes, including RouteID, StartLocation, and EndLocation from the "BusRoute" table.

3. `v_ticket_types`: Fetches details of different ticket types, such as TicketTypeID, TypeName, Price, ValidityPeriod, and Description from the "TicketType" table.

4. `v_user_update`: Gathers user information, excluding the Date of Birth, from the "User" table, including UserId, FirstName, LastName, Email, Phone Number, and Username.

5. `v_payment_info`: Combines payment data with user and ticket type details, providing PaymentID, Amount, PaymentDate, user's FirstName, LastName, Username, and the TypeName of the associated ticket.

6. `v_schedule`: Retrieves information about bus schedules, including ScheduleID, DayFlag, ArrivalTime, and DepartureTime from the "Schedule" table.

7. `v_bus_stops`: Fetches data about bus stops, including StopID, StopName, and Location from the "BusStop" table.

8. `v_route_stop_seq`: Joins route stop sequence data with bus stop details, showing the NumberofStops, ArrivalTime, StopName, and Location.

9. `v_bus_schedule`: Combines information about buses, their schedules, and associated routes, including BusID, BusNumber, ScheduleID, DayFlag, ArrivalTime, and DepartureTime.

10. `BusScheduleView`: A view that combines various aspects of bus schedules, including BusNumber, ArrivalTime, DepartureTime, NumberofStops, StartLocation, EndLocation, and DayFlag. This view likely provides a comprehensive overview of bus schedules, routes, and stops.

- After creating these views, the code checks for the existence of the "BusScheduleView" and drops it if it already exists. Finally, it selects data from the "BusScheduleView" based on specific filtering conditions for StartLocation, EndLocation, and DayFlag.
- The code seems to be related to a database system managing bus transportation information, with the views simplifying the retrieval of various pieces of information. The last SELECT statement is intended to retrieve specific data from the "BusScheduleView" based on certain criteria, which can be customized by replacing 'YourStartLocation,' 'YourEndLocation,' and 'YourDayFlag' with specific values.

10.2.1 Code of View Implementation

```
CREATE VIEW v_user_info AS
SELECT UserId, FirstName, LastName, DOB, Email, PhoneNumber, Username
FROM User;

CREATE VIEW v_bus_routes AS
SELECT RouteID, StartLocation, EndLocation
FROM BusRoute;

CREATE VIEW v_ticket_types AS
SELECT TicketTypeID, TypeName, Price, ValidityPeriod, Description
FROM TicketType;

CREATE VIEW v_user_update AS
SELECT UserId, FirstName, LastName, Email, PhoneNumber, Username
FROM User;

CREATE VIEW v_payment_info AS
SELECT p.PaymentID, p.Amount, p.PaymentDate,
    u.FirstName, u.LastName, u.Username,
    t.TypeName
FROM Payment p
JOIN User u ON p.User_UserId = u.UserId
JOIN TicketType t ON p.TicketType_TicketTypeId = t.TicketTypeId;
select *from v_bus_schedule;

CREATE VIEW v_schedule AS
SELECT ScheduleID, DayFlag, ArrivalTime, DepartureTime
FROM Schedule;

CREATE VIEW v_bus_stops AS
SELECT StopID, StopName, Location
FROM BusStop;

CREATE VIEW v_route_stop_seq AS
SELECT rs.NumberofStops, rs.ArrivalTime,
    bs.StopName, bs.Location
FROM RouteStopSequence rs
JOIN BusStop bs ON rs.BusStop_StopID = bs.StopID;

CREATE VIEW v_bus_schedule AS
SELECT b.BusID, b.BusNumber, s.ScheduleID, s.DayFlag, s.ArrivalTime, s.DepartureTime
FROM Bus_has_Schedule bs
JOIN Bus b ON bs.Bus_BusID = b.BusID
JOIN Schedule s ON bs.Schedule_ScheduleID = s.ScheduleID;

CREATE VIEW v_bus_routes AS
SELECT b.BusID, b.BusNumber, r.RouteID, r.StartLocation, r.EndLocation
FROM Bus_has_BusRoute bbr
JOIN Bus b ON bbr.Bus_BusID = b.BusID
JOIN BusRoute r ON bbr.BusRoute_RouteID = r.RouteID;

SHOW FULL TABLES IN dutchess_county_bus_transportation_dbms_project WHERE
```

```
TABLE_TYPE = 'VIEW';

CREATE VIEW BusScheduleView AS
SELECT b.BusNumber,
       s.ArrivalTime,
       s.DepartureTime,
       rs.NumberofStops,
       r.StartLocation,
       r.EndLocation,
       s.DayFlag
FROM Schedule s
JOIN Bus_has_Schedule bs ON s.ScheduleID = bs.Schedule_ScheduleID
JOIN Bus b ON bs.Bus_BusID = b.BusID
JOIN Bus_has_BusRoute bbr ON b.BusID = bbr.Bus_BusID
JOIN BusRoute r ON bbr.BusRoute_RouteID = r.RouteID
JOIN BusRoute r ON bbr.BusRoute_RouteID = r.RouteID
JOIN RouteStopSequence rs ON r.RouteID = rs.BusStop_StopID;

DROP VIEW IF EXISTS BusScheduleView;
SELECT *FROM BusScheduleView;



SELECT * FROM BusScheduleView WHERE StartLocation = 'YourStartLocation' AND
EndLocation = 'YourEndLocation' AND DayFlag = 'YourDayFlag';
```

11. GRAPHICAL USER INTERFACE DESIGN

a.Database Configuration

This Python code is a simple GUI application using the Tkinter library for creating a login window. The application connects to a MySQL database and displays an image fetched from a URL using the requests library and then processed with the Pillow (PIL) library. Let me break down the code:

The Tkinter library is imported for creating the graphical user interface (GUI) components, and specific modules like ttk and messagebox are imported for additional UI elements.

The MySQL connector library is imported to establish a connection with a MySQL database. The connection details are specified in the 'config' dictionary, including the host, user, password, and database name.

The root window is created using Tkinter, and its title is set to "Login."

A global variable 'login_image_tk' is initialized to None. This variable will be used to store the image data retrieved from the URL.

The script establishes a connection to the MySQL database using the provided configuration and creates a cursor to execute SQL queries.

The image for the login window is fetched from a URL using the requests library. The image data is then saved to a file named "debug_image.jpg" for debugging purposes.

The code attempts to open the image using the Pillow library, convert it to a Tkinter-compatible format (PhotoImage), and display it in the Tkinter window. If successful, a label widget ('image_label') is created, and the image is packed into the root window. An exception is caught and printed if any error occurs during this process.

It's worth noting that the image URL provided seems to be incorrect. The URL points to an HTML page at "https://ibb.co/QM5B7Dg" rather than a direct image file. This might cause issues in fetching and displaying the image correctly.

Additionally, the code establishes a connection to the database, but it doesn't perform any specific database operations in this snippet. It might be part of a larger application where the user would input credentials in the GUI, and the script would authenticate against the MySQL database.

CODE :

```
import tkinter as tk
from tkinter import ttk, messagebox
import mysql.connector
from datetime import timedelta
from PIL import Image, ImageTk
import requests
from io import BytesIO
```

```
# Make login_image_tk a global variable
login_image_tk = None

# Database config
config = {
    "host": "localhost",
    "user": "root",
    "password": "shannu4321",
    "database": "dutchess_county_bus_transportation_dbms_project"
}

# Root window
root = tk.Tk()
root.title("Login")

# Database connection and cursor
db = mysql.connector.connect(**config)
cursor = db.cursor()

# Load and display login image
image_url = "https://ibb.co/QM5B7Dg"
image_data = requests.get(image_url).content


# Try saving the content to a file (debugging purposes)
with open("debug_image.jpg", "wb") as f:
    f.write(image_data)

try:
    login_image = Image.open(BytesIO(image_data))
    login_image_tk = ImageTk.PhotoImage(login_image)
    image_label = ttk.Label(root, image=login_image_tk)
    image_label.image = login_image_tk  # Keep a reference to the image
    image_label.pack()
except Exception as e:
    print("Error:", e)
```

b.Login Page

The GUI seems to represent a basic authentication system with a login page, sign-up functionality, and a password recovery feature.

The login page consists of entry fields for a username and password, and upon clicking the "Login" button, it attempts to authenticate the user by querying a database using the provided credentials. If the authentication is successful, a success message is shown, and the `main_menu()` function is called. If unsuccessful, an error message is displayed.

The code also includes a "Forgot Password?" button that, when clicked, opens a new window allowing the user to reset their password. The password reset functionality involves updating the password in the database.

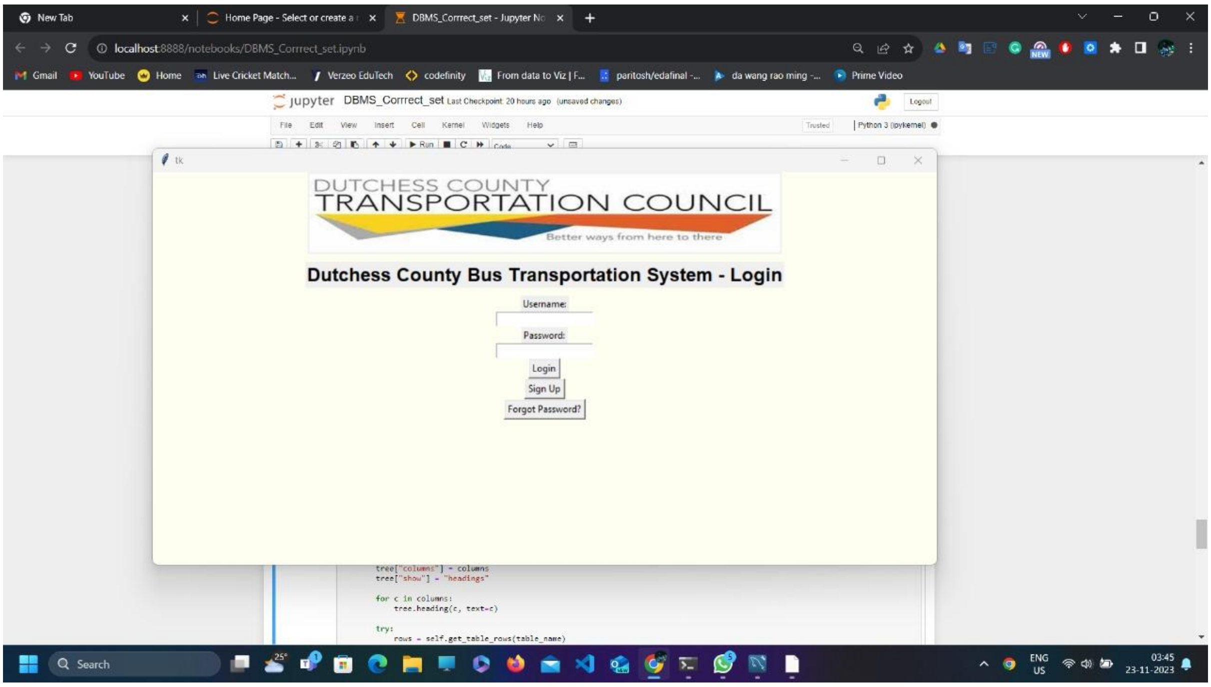
Additionally, there is a "Sign Up" button that opens a new window for user registration. The sign-up form collects information such as first name, last name, email, username, and password. Upon submitting the sign-up form, the user's information is inserted into the database, and a success message is displayed.

It's important to note that the code references a database and assumes the existence of a cursor and connection (`db`). The specifics of the database schema and the `main_menu()` function are not provided in this snippet. Also, there are some global variables that should be used with caution, as global variables can introduce complexity and potential issues in larger codebases.

CODE:

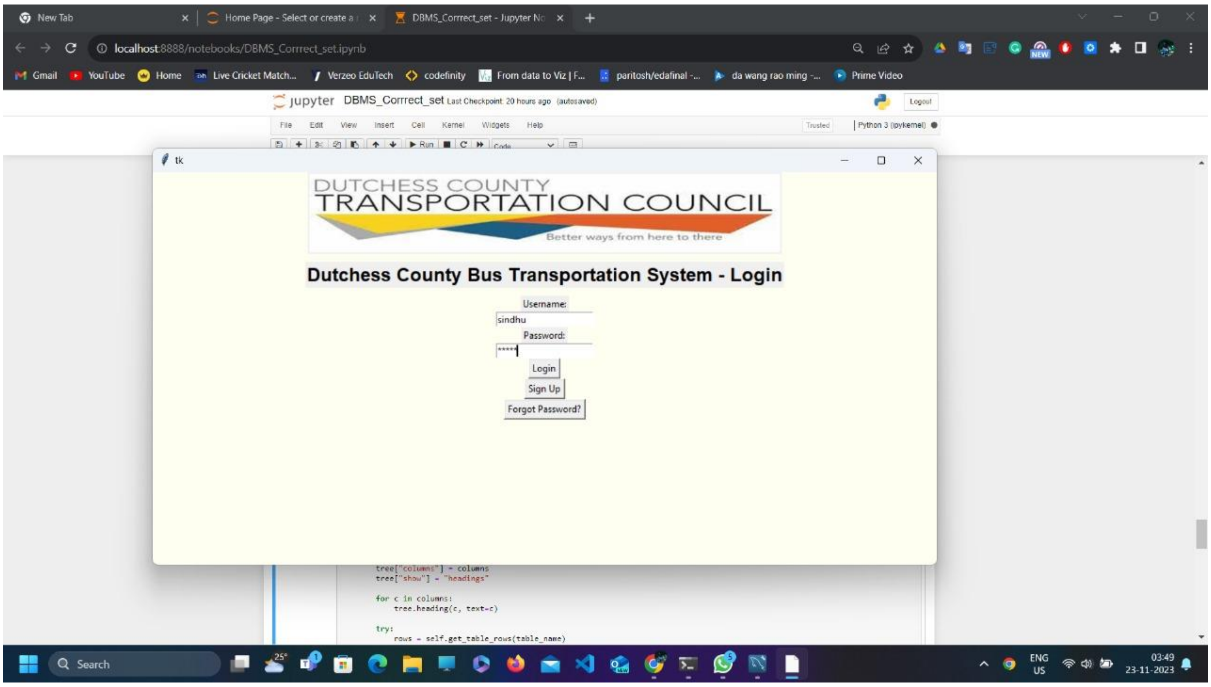
```
# Login page
username_label = tk.Label(root, text="Username:")
username_entry = tk.Entry(root)

password_label = tk.Label(root, text="Password:")
password_entry = tk.Entry(root, show='*')

def login():
    username = username_entry.get()
    password = password_entry.get()

    # Check credentials in v_user_info view
    cursor.execute("SELECT * FROM v_user_info WHERE username=%s AND password=%s",
(username, password))

    if cursor.fetchone() is None:
        messagebox.showerror("Error","Invalid credentials")
    else:
        messagebox.showinfo("Success","Login successful")
        main_menu()

# Forgot password
# Make entry variables global
forgot_username_entry = None
forgot_password_entry = None
forgot_window = None
# ...

# Forgot password
def forgot_password():
    global forgot_username_entry, forgot_password_entry
    forgot_window = tk.Toplevel(root)

    forgot_username_label = tk.Label(forgot_window, text="Username:")
    forgot_username_entry = tk.Entry(forgot_window)

    forgot_password_label = tk.Label(forgot_window, text="New Password:")
    forgot_password_entry = tk.Entry(forgot_window, show='*')

    forgot_btn = tk.Button(forgot_window, text="Reset Password", command=forgot_submit)
    # Pack widgets
    forgot_username_label.pack()
    forgot_username_entry.pack()

    forgot_password_label.pack()
    forgot_password_entry.pack()

    forgot_btn.pack()

# Forgot password submit
def forgot_submit():
    global forgot_username_entry, forgot_password_entry, forgot_window
    username = forgot_username_entry.get()
    new_password = forgot_password_entry.get()

    # Update password in v_user_info view
    update_sql = "UPDATE v_user_info SET password = %s WHERE username = %s"
    cursor.execute(update_sql, (new_password, username))

    db.commit()

    messagebox.showinfo("Success", "Password reset successfully!")

    # Check if forgot_window exists before destroying
    if forgot_window:
        forgot_window.destroy()


# Make entry variables global
firstname_entry = None
lastname_entry = None
email_entry = None
username_entry2 = None
password_entry2 = None

# ...

# Signup
# Signup
def signup():
    global firstname_entry, lastname_entry, email_entry, username_entry2, password_entry2
    signup_window = tk.Toplevel(root)

    # Labels and entries for each field
    firstname_label = tk.Label(signup_window, text="First Name:")
    firstname_entry = tk.Entry(signup_window)

    lastname_label = tk.Label(signup_window, text="Last Name:")
    lastname_entry = tk.Entry(signup_window)

    email_label = tk.Label(signup_window, text="Email:")
    email_entry = tk.Entry(signup_window)

    username_label = tk.Label(signup_window, text="Username:")
    username_entry2 = tk.Entry(signup_window)
    password_label = tk.Label(signup_window, text="Password:")
    password_entry2 = tk.Entry(signup_window, show='*')

    signup_btn = tk.Button(signup_window, text="Sign Up", command=signup_submit)

    # Pack the widgets
    firstname_label.pack()
    firstname_entry.pack()

    lastname_label.pack()
    lastname_entry.pack()

    email_label.pack()
    email_entry.pack()

    username_label.pack()
    username_entry2.pack()

    password_label.pack()
    password_entry2.pack()

    signup_btn.pack()

# Signup submit
def signup_submit():
    global firstname_entry, lastname_entry, email_entry, username_entry, password_entry
    first_name = firstname_entry.get()
    last_name = lastname_entry.get()
    email = email_entry.get()
    username = username_entry.get()
    password = password_entry.get()

    # Insert with values for all columns
    insert_sql = "INSERT INTO v_user_info (FirstName, LastName, Email, Username, Password)
VALUES (%s, %s, %s, %s, %s)"
    values = (first_name, last_name, email, username, password)

    cursor.execute(insert_sql, values)

    # Rest of signup
    db.commit()

    messagebox.showinfo("Success", "Account created!")


login_btn = tk.Button(root, text="Login", command=login)
signup_btn = tk.Button(root, text="Sign Up", command=signup)
forgot_pass_btn = tk.Button(root, text="Forgot Password?", command=forgot_password)

# Pack widgets
username_label.pack()
username_entry.pack()

password_label.pack()
```

```
password_entry.pack()

login_btn.pack()
signup_btn.pack()
forgot_pass_btn.pack()
```


localhost:8888/notebooks/DBMS_Correct_set.ipynb
jupyter DBMS_Corrrect_set Last Checkpoint: 20 hours ago (unsaved changes)
tk
DUTCHESS COUNTY
TRANSPORTATION COUNCIL
Better ways from here to there
Dutchess County Bus Transportation System - Login
Username:
Password:
Login
Sign Up
Forgot Password?

```
tree["columns"] = columns
tree["show"] = "headings"

for c in columns:
    tree.heading(c, text=c)

try:
    rows = self.get_table_rows(table_name)
```

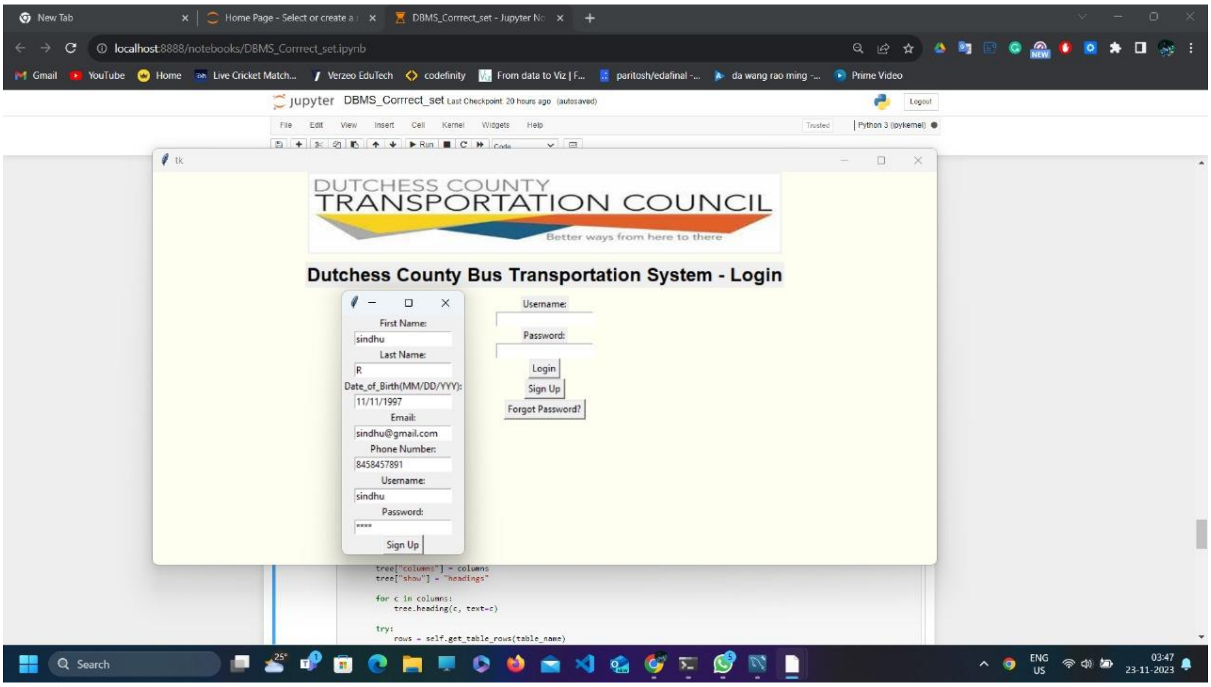
Figure 11. 1 Landing Page


localhost:8888/notebooks/DBMS_Correct_set.ipynb
jupyter DBMS_Corrrect_set Last Checkpoint: 20 hours ago (autosaved)
tk
DUTCHESS COUNTY
TRANSPORTATION COUNCIL
Better ways from here to there
Dutchess County Bus Transportation System - Login
Login
Sign Up
Forgot Password?

Username: sindhu
Password: *****

```
tree["columns"] = columns
tree["show"] = "headings"

for c in columns:
    tree.heading(c, text=c)

try:
    rows = self.get_table_rows(table_name)
```

Figure 11. 2 Login Page


localhost:8888/notebooks/DBMS_Corrrect_set.ipynb
jupyter DBMS_Corrrect_set Last Checkpoint: 20 hours ago (autosaved)
DUTCHESS COUNTY
TRANSPORTATION COUNCIL
Better ways from here to there
Login
Sign Up
Forgot Password?
Sign Up

Dutchess County Bus Transportation System - Login

First Name: sindhu

Last Name: R

Date_of_Birth(MM/DD/YYY): 11/11/1997

Email: sindhu@gmail.com

Phone Number: 8458457891

Username: sindhu

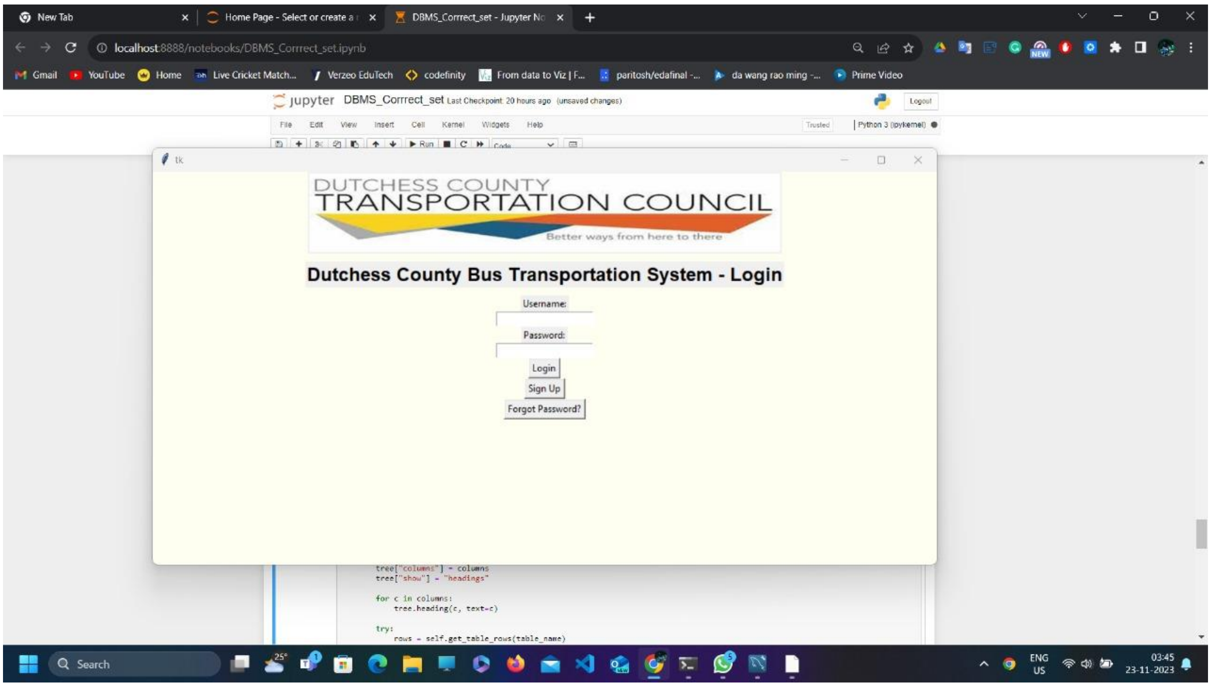
Password: ****

Username:

Password:

```
    tree["columns"] = columns
    tree["show"] = "headings"

    for c in columns:
        tree.heading(c, text=c)

    try:
        rows = self.get_table_rows(table_name)
```

Figure 11. 3 Signup Page


localhost:8888/notebooks/DBMS_Corrrect_set.ipynb
jupyter DBMS_Corrrect_set Last Checkpoint: 20 hours ago (unsaved changes)
DUTCHESS COUNTY
TRANSPORTATION COUNCIL
Better ways from here to there
Login
Sign Up
Forgot Password?

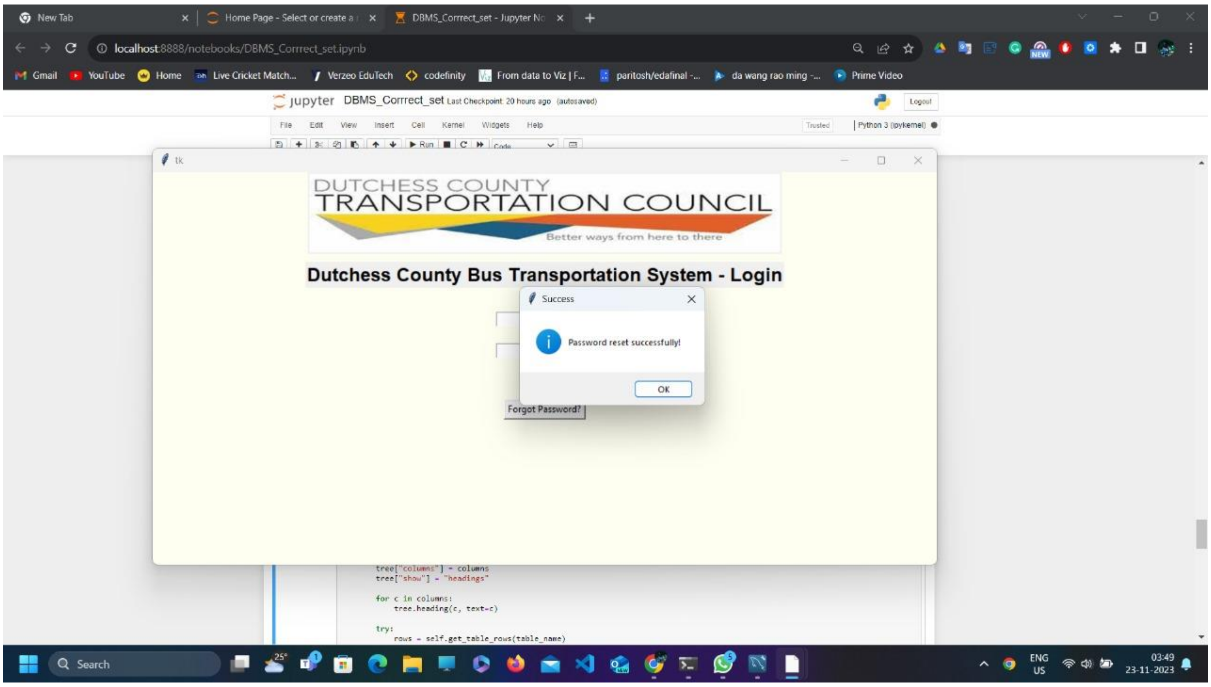
Dutchess County Bus Transportation System - Login

Username:

Password:

```
    tree["columns"] = columns
    tree["show"] = "headings"

    for c in columns:
        tree.heading(c, text=c)

    try:
        rows = self.get_table_rows(table_name)
```

Figure 11. 4 Successful login page



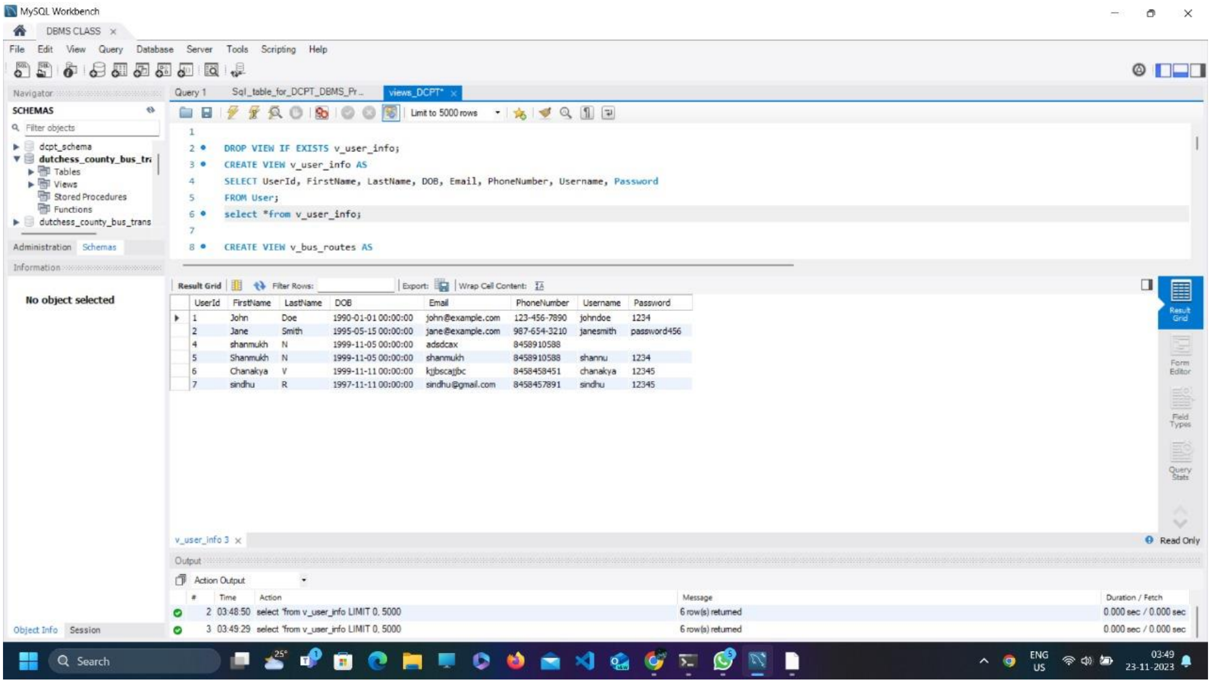
jupyter DBMS_Corrrect_set Last Checkpoint: 20 hours ago (autosaved)
DUTCHESS COUNTY TRANSPORTATION COUNCIL
Better ways from here to there
Success
OK
Forgot Password?

Dutchess County Bus Transportation System - Login

Password reset successfully!

```
tree["columns"] = columns
tree["show"] = "headings"

for c in columns:
    tree.heading(c, text=c)

try:
    rows = self.get_table_rows(table_name)
```

Figure 11. 5 Password Reset

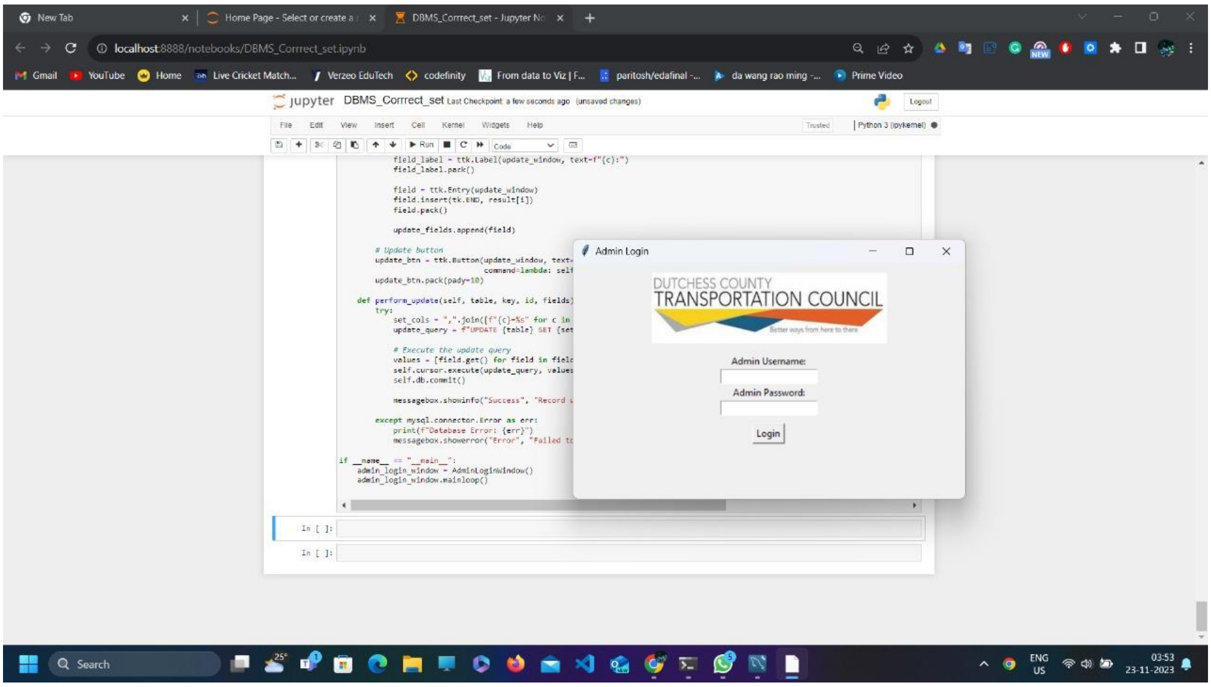


```
DROP VIEW IF EXISTS v_user_info;
CREATE VIEW v_user_info AS
SELECT UserId, FirstName, LastName, DOB, Email, PhoneNumber, Username, Password
FROM User;
select *from v_user_info;

CREATE VIEW v_bus_routes AS
```

UserId	FirstName	LastName	DOB	Email	PhoneNumber	Username	Password
1	John	Doe	1990-01-01 00:00:00	john@example.com	123-456-7890	johndoe	1234
2	Jane	Smith	1995-05-15 00:00:00	jane@example.com	987-654-3210	janesmith	password456
4	shanmukh	N	1999-11-05 00:00:00	adsdcax	8458910588		
5	Shanmukh	N	1999-11-05 00:00:00	shanmukh	8458910588	shannu	1234
6	Chanakya	V	1999-11-11 00:00:00	kjbscajbc	8458458451	chanakya	12345
7	sindhu	R	1997-11-11 00:00:00	sindhu@gmail.com	8458457891	sindhu	12345

```
#  Time      Action                                  Message             Duration / Fetch
2  03:48:50  select *from v_user_info LIMIT 0, 5000  6 row(s) returned   0.000 sec / 0.000 sec
3  03:49:29  select *from v_user_info LIMIT 0, 5000  6 row(s) returned   0.000 sec / 0.000 sec
```

Figure 11. 6 Password update in Database



```
            field_label = ttk.Label(update_window, text=f"{c}:")
            field_label.pack()

            field = ttk.Entry(update_window)
            field.insert(tk.END, result[i])
            field.pack()

            update_fields.append(field)

        # Update button
        update_btn = ttk.Button(update_window, text-
                                command=lambda: self
        update_btn.pack(pady=10)

    def perform_update(self, table, key, id, fields)
        try:
            set_cols = ",".join([f"{c}=%s" for c in
            update_query = f"UPDATE {table} SET {set

            # Execute the update query
            values = [field.get() for field in fielc
            self.cursor.execute(update_query, values
            self.db.commit()

            messagebox.showinfo("Success", "Record u

        except mysql.connector.Error as err:
            print(f"Database Error: {err}")
            messagebox.showerror("Error", "Failed tc

if __name__ == "__main__":
    admin_login_window = AdminLoginWindow()
    admin_login_window.mainloop()
```

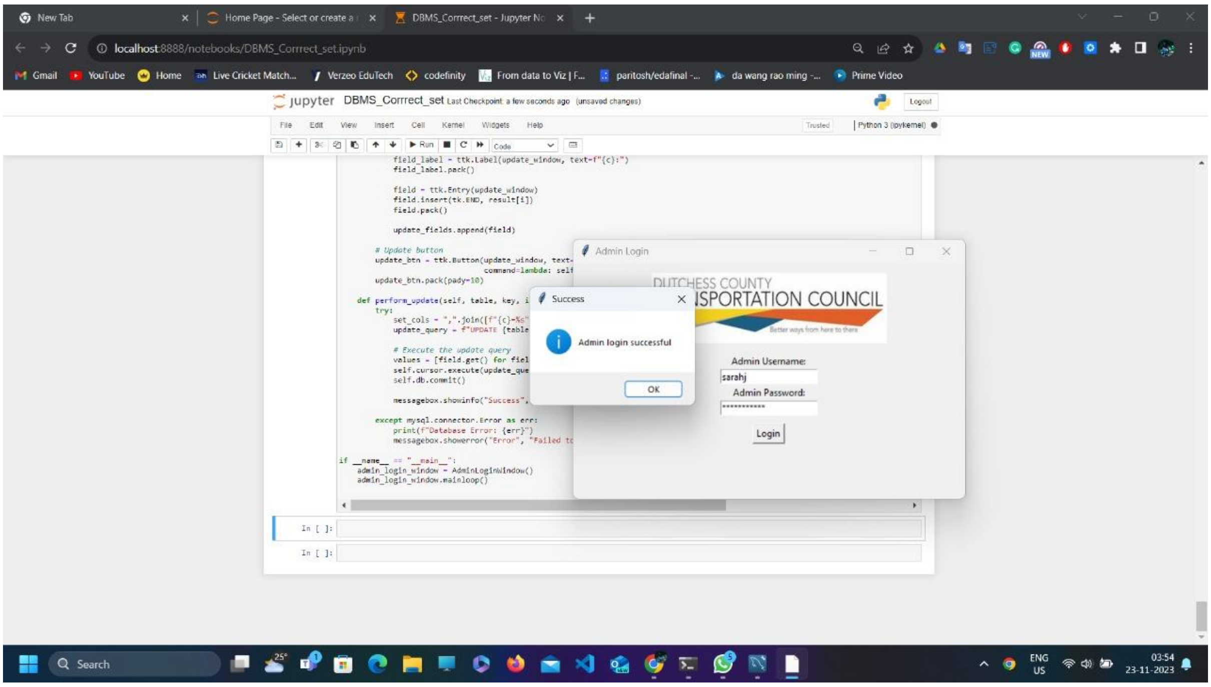
In [ ]:

In [ ]:

Admin Username:

Admin Password:

Figure 11. 7 Admin Login Page


New Tab
Home Page - Select or create a
DBMS_Corrrect_set - Jupyter No
localhost:8888/notebooks/DBMS_Corrrect_set.ipynb
jupyter
DBMS_Corrrect_set
Last Checkpoint: a few seconds ago (unsaved changes)
Logout
Run
Code
Admin Login
DUTCHESS COUNTY
TRANSPORTATION COUNCIL
Better ways from here to there
Success
OK
Login
Search
ENG
US
03:54
23-11-2023

Admin login successful

Admin Username: sarahj

Admin Password: ***********

Figure 11. 8 Admin Login Successful

c.Main Menu Page

The provided code defines a function called `main_menu` that serves as the main interface for a transportation scheduling and payment system. The function begins by clearing the existing widgets in the root window, effectively serving as a transition from a login or previous screen to the main menu. It then sets the title of the window to "Main Menu."

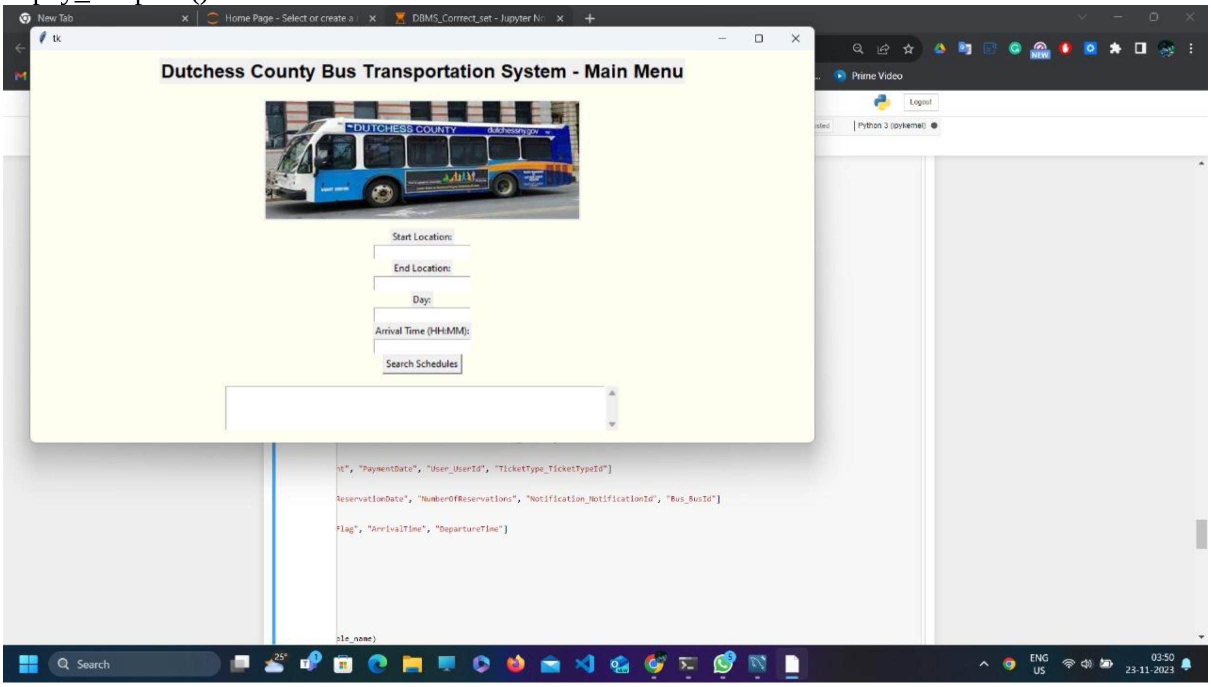
The main menu consists of several entry fields, labels, and buttons created using the Tkinter library in Python. Entry widgets are used for user input, including start location, end location, day, and arrival time. These input fields are accompanied by corresponding labels for clarity. The function also declares global variables (`start_entry`, `end_entry`, `day_entry`, and `arrival_time_entry`) to make these entry widgets accessible outside the scope of the function. This is useful for retrieving user input values in other parts of the program.

Two buttons are included in the main menu: "Search Schedules" and "Make Payment." These buttons are linked to the functions `search` and `make_payment` respectively, suggesting that the main menu is a gateway for users to search for transportation schedules and proceed with payment transactions. Overall, this code establishes the graphical user interface (GUI) for the transportation system, enabling users to input relevant information and interact with the system's functionalities.

CODE :

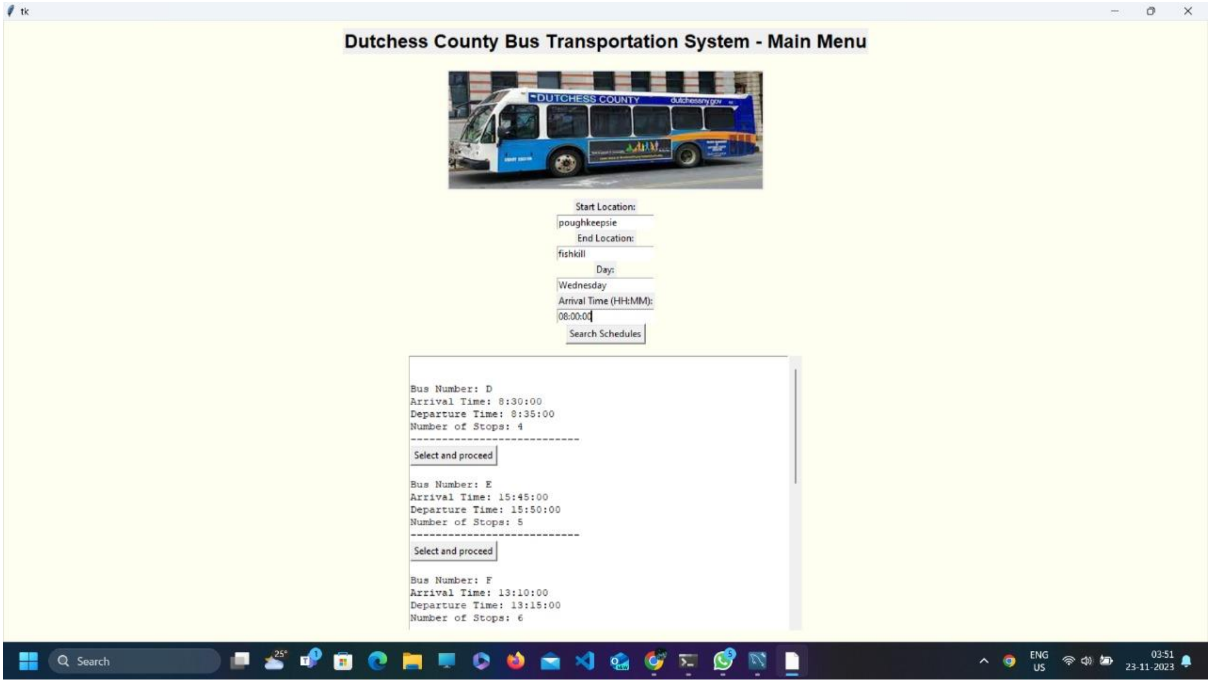
```
# Main menu
def main_menu():
    global start_entry, end_entry, day_entry, arrival_time_entry  # Declare as global variables

    # Clear the login window
    for widget in root.winfo_children():
        widget.destroy()

    root.title("Main Menu")

    # Create Entry widgets for user input
    start_label = tk.Label(root, text="Start Location:")
    start_entry = tk.Entry(root)

    end_label = tk.Label(root, text="End Location:")
    end_entry = tk.Entry(root)

    day_label = tk.Label(root, text="Day:")
    day_entry = tk.Entry(root)

    arrival_time_label = tk.Label(root, text="Arrival Time (HH:MM):")
    arrival_time_entry = tk.Entry(root)  # Create an entry for arrival time

    start_label.pack()
    start_entry.pack()

    end_label.pack()
    end_entry.pack()

    day_label.pack()
    day_entry.pack()

    arrival_time_label.pack()
    arrival_time_entry.pack()  # Pack the arrival time entry

    search_btn = tk.Button(root, text="Search Schedules", command=search)
    search_btn.pack()
```

```
pay_btn = tk.Button(root, text="Make Payment", command=make_payment)
pay_btn.pack()
```



```
nt", "PaymentDate", "User_UserId", "TicketType_TicketTypeId"]

ReservationDate", "NumberOfReservations", "Notification_NotificationId", "Bus_BusId"]

Flag", "ArrivalTime", "DepartureTime"]

ble_name)
```

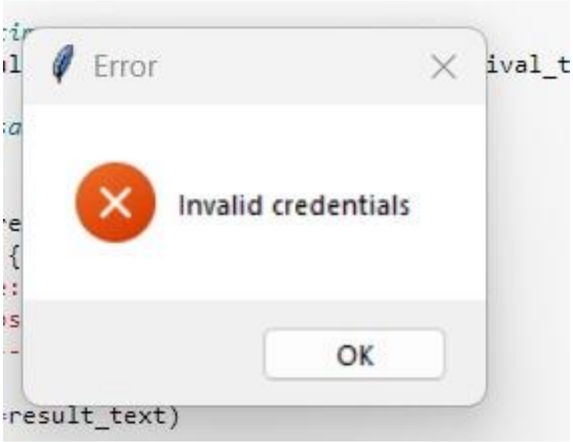
Figure 11. 9 Main Menu Page



Start Location: poughkeepsie
End Location: fishkill
Day: Wednesday
Arrival Time (HH:MM): 08:00:00

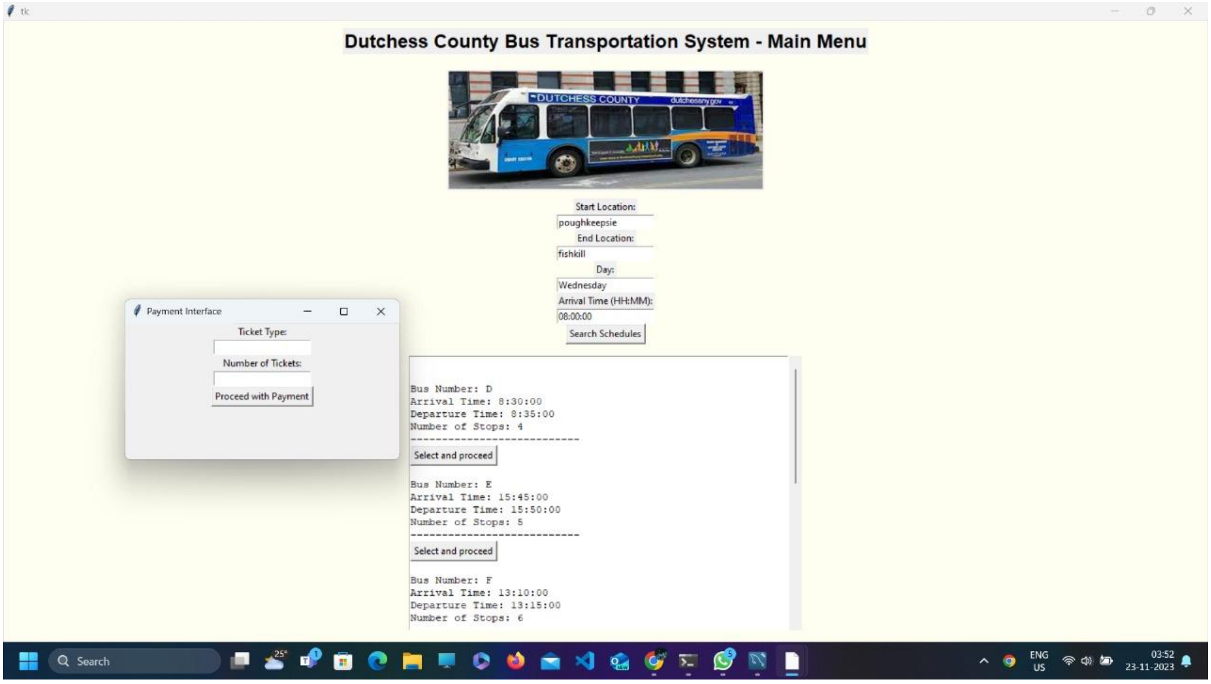
```
Bus Number: D
Arrival Time: 8:30:00
Departure Time: 8:35:00
Number of Stops: 4
---------------------------

Bus Number: E
Arrival Time: 15:45:00
Departure Time: 15:50:00
Number of Stops: 5
---------------------------

Bus Number: F
Arrival Time: 13:10:00
Departure Time: 13:15:00
Number of Stops: 6
```

Figure 11. 10 Main Menu Page after entering Details



Invalid credentials

Figure 11. 11 Invalid Credentials



Dutchess County Bus Transportation System - Main Menu

Start Location: poughkeepsie

End Location: fishkill

Day: Wednesday

Arrival Time (HH:MM): 08:00:00

Ticket Type:

Number of Tickets:

```
Bus Number: D
Arrival Time: 8:30:00
Departure Time: 8:35:00
Number of Stops: 4
---------------------------

Bus Number: E
Arrival Time: 15:45:00
Departure Time: 15:50:00
Number of Stops: 5
---------------------------

Bus Number: F
Arrival Time: 13:10:00
Departure Time: 13:15:00
Number of Stops: 6
```

Figure 11. 12 Ticket Page

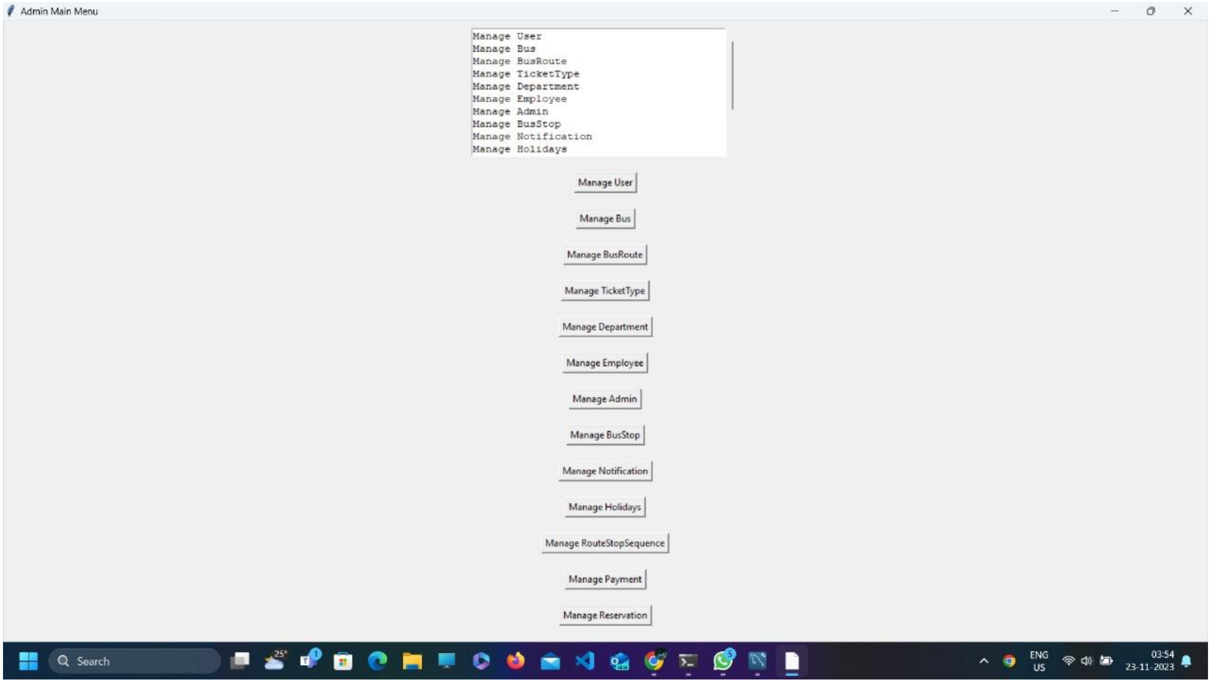
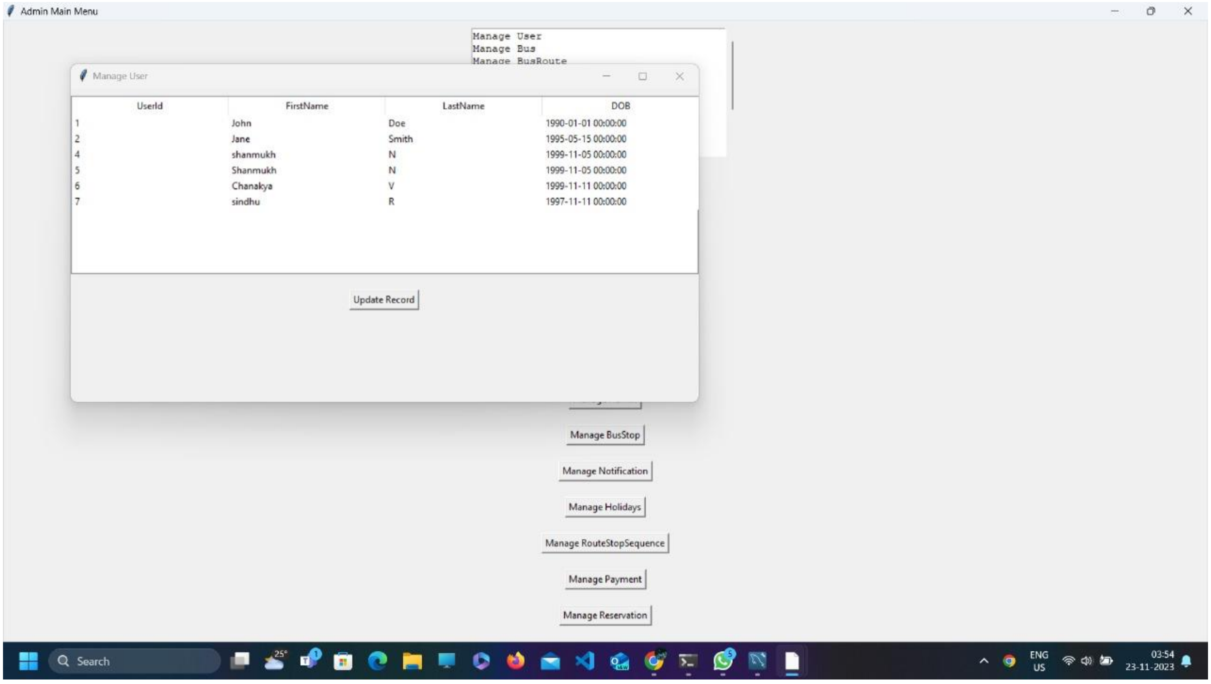
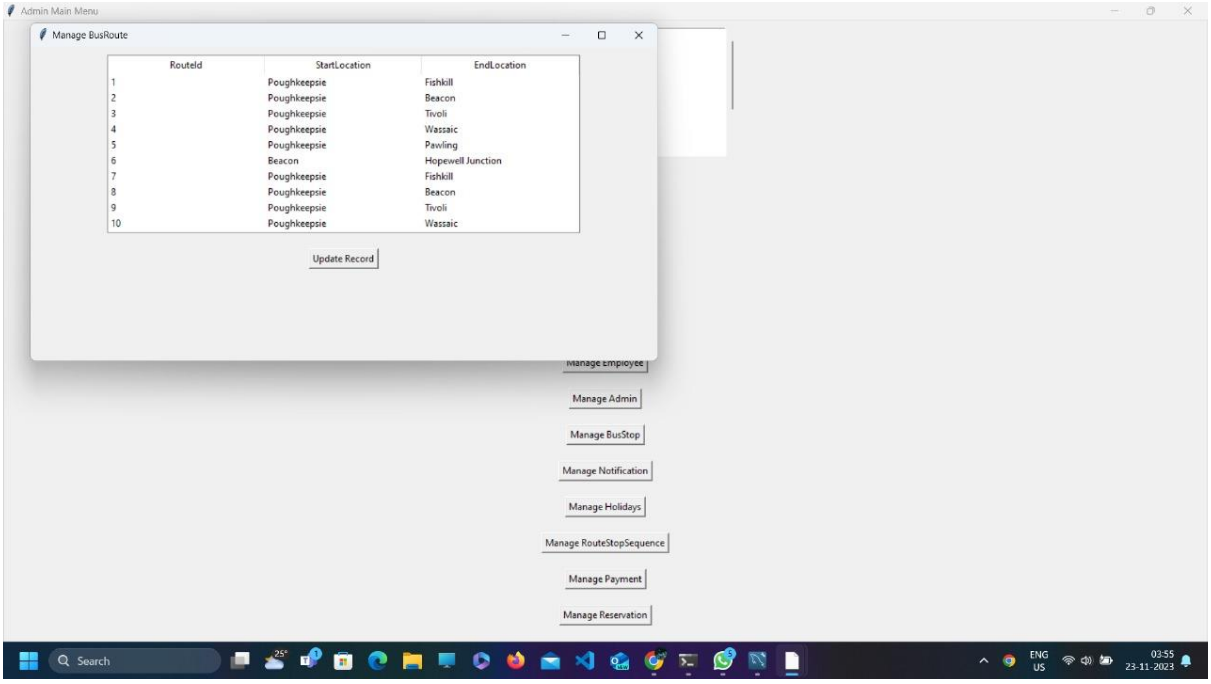


Figure 11. 13 Admin Menu



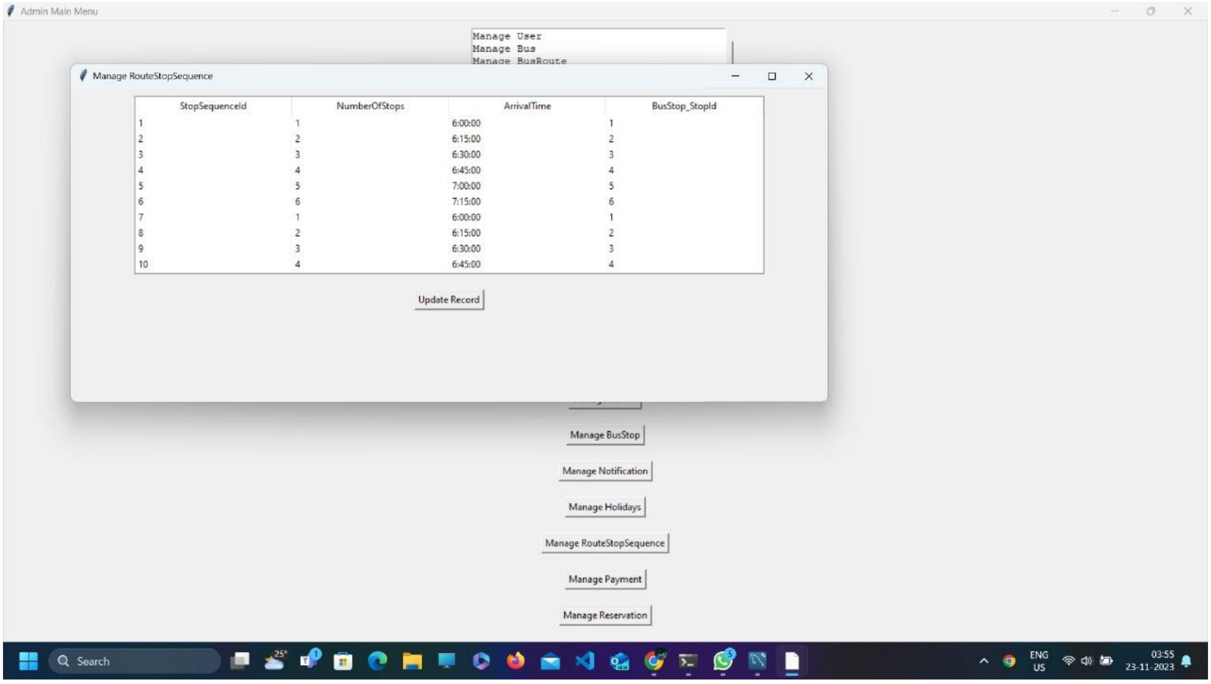
UserId	FirstName	LastName	DOB
1	John	Doe	1990-01-01 00:00:00
2	Jane	Smith	1995-05-15 00:00:00
4	shanmukh	N	1999-11-05 00:00:00
5	Shanmukh	N	1999-11-05 00:00:00
6	Chanakya	V	1999-11-11 00:00:00
7	sindhu	R	1997-11-11 00:00:00

Figure 11. 14 Admin managing User



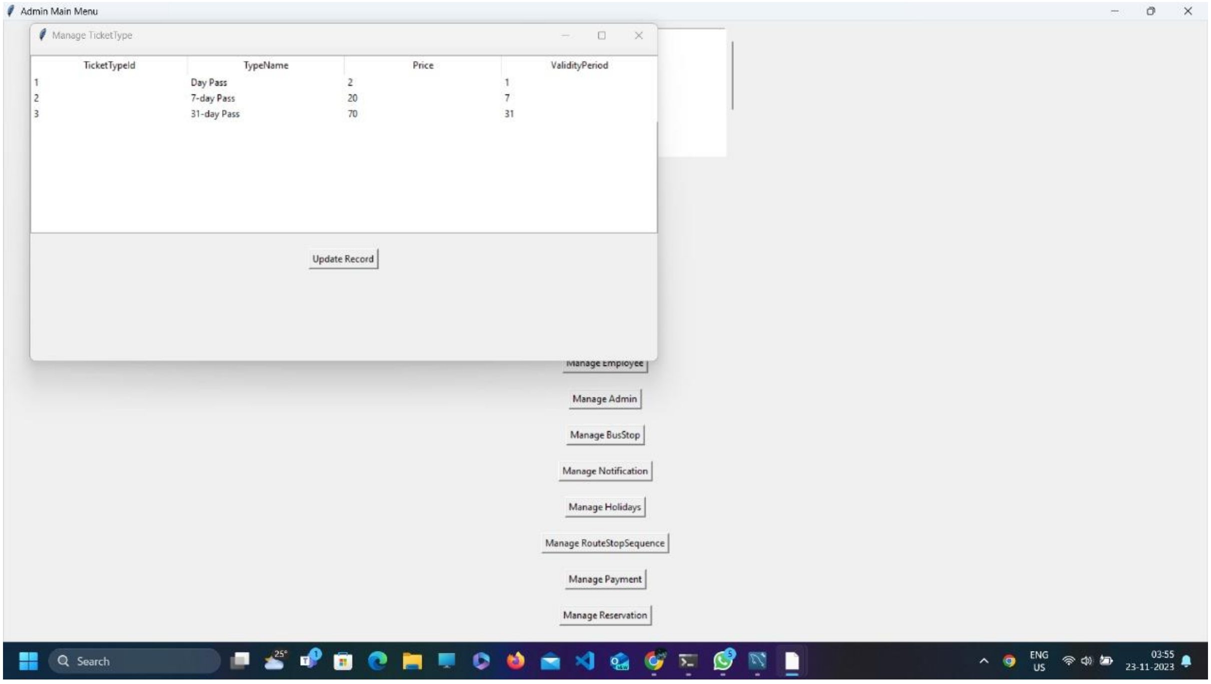
RouteId	StartLocation	EndLocation
1	Poughkeepsie	Fishkill
2	Poughkeepsie	Beacon
3	Poughkeepsie	Tivoli
4	Poughkeepsie	Wassaic
5	Poughkeepsie	Pawling
6	Beacon	Hopewell Junction
7	Poughkeepsie	Fishkill
8	Poughkeepsie	Beacon
9	Poughkeepsie	Tivoli
10	Poughkeepsie	Wassaic

Figure 11. 15 Admin managing BusRoute



StopSequenceId	NumberOfStops	ArrivalTime	BusStop_StopId
1	1	6:00:00	1
2	2	6:15:00	2
3	3	6:30:00	3
4	4	6:45:00	4
5	5	7:00:00	5
6	6	7:15:00	6
7	1	6:00:00	1
8	2	6:15:00	2
9	3	6:30:00	3
10	4	6:45:00	4

Figure 11. 16 Admin Managing Route



TicketTypeId	TypeName	Price	ValidityPeriod
1	Day Pass	2	1
2	7-day Pass	20	7
3	31-day Pass	70	31

Figure 11. 17 Admin managing ticket type

d.Action Page

- **Search Page**

The `search` function seems to be the core of the application, where user input is gathered to search for bus schedules based on criteria such as start and end locations, day, and arrival time. The code establishes a connection to a database, executes a query to retrieve bus schedules, filters the results based on the user's specified arrival time, and then displays the relevant information in the GUI.

To enhance the application, it is suggested to implement additional functionalities such as image integration, user account creation, password recovery, and login capabilities. This could involve creating separate functions for handling these tasks, and incorporating corresponding UI elements like image display widgets, entry fields for user registration, login credentials, and buttons for account-related actions. Furthermore, it's crucial to secure the login functionality and handle user authentication securely. Additionally, implementing a payment feature, as indicated by the placeholder comment in the `make_payment` function, could involve integrating a payment gateway or method.

To improve the code's structure, it's advisable to encapsulate related functionalities into classes and methods, promoting a modular and organized design. Additionally, error handling and user feedback can be enhanced to provide more informative messages in case of database errors or other issues. Overall, the code provides a foundation for a bus schedule application but would benefit from additional features and refinements for a more comprehensive and user-friendly experience.

CODE :

```
# Search
```

```
def search():
    # Get user input
    start = start_entry.get()
    end = end_entry.get()
    day = day_entry.get()
    arrival_time_str = arrival_time_entry.get()

    # Convert the user input arrival time to a timedelta object
    arrival_time = timedelta(hours=int(arrival_time_str.split(':')[0]),
minutes=int(arrival_time_str.split(':')[1]))

    # Database connection and search functionality
    try:
        # Construct and execute the database query
        cursor.execute("SELECT * FROM BusScheduleView")
        results = cursor.fetchall()

        # Filter results based on arrival time
        filtered_results = [result for result in results if result[1] >= arrival_time]

        # Display filtered results in UI (sample implementation)
        result_text = ""
        for result in filtered_results:
            result_text += f"Bus Number: {result[0]}\n"
            result_text += f"Arrival Time: {result[1]}\n"
            result_text += f"Departure Time: {result[2]}\n"
            result_text += f"Number of Stops: {result[3]}\n"
            result_text += "---------------------------\n"

        result_label = tk.Label(root, text=result_text)
result_label.pack()

    except mysql.connector.Error as err:
        print(f"Database Error: {err}")

# Make payment
def make_payment():
    # Payment page functionality
    pass

root.mainloop()
```

insertion values

```
        # Insert with values for all columns
        insert_sql = "INSERT INTO User (FirstName, LastName, DOB, Email, PhoneNumber,
Username, Password) VALUES (%s, %s, %s, %s, %s, %s, %s)"
        values = (first_name, last_name, dob, email, phone_number, username, password)

        cursor.execute(insert_sql, values)

        db.commit()
        messagebox.showinfo("Success", "Account created!")

    except ValueError as e:
        messagebox.showerror("Error", f"Invalid date format for Date of Birth: {dob_str}. Please use 
MM/DD/YYYY.")
```

- **Modification page**

The provided code appears to be a part of a password recovery or reset functionality in a graphical user interface (GUI) application. It defines a function named `forgot_submit`, which presumably gets called when a user submits a password reset request. The function retrieves the entered username and new password from GUI entry fields (presumably in a Tkinter-based interface, given the use of `messagebox`). It then performs a SQL UPDATE operation on a database table named `v_user_info`, setting the new password for the specified username. After successfully updating the database, it displays a success message using a messagebox. Additionally, it checks if a window named `forgot_window` exists and, if so, destroys it. It's important to note that the security implications of this code depend on the broader context, such as how user inputs are validated, how passwords are stored, and whether SQL injection vulnerabilities are adequately addressed.

CODE:

```
# Forgot password submit
def forgot_submit():
    global forgot_username_entry, forgot_password_entry, forgot_window
    username = forgot_username_entry.get()
    new_password = forgot_password_entry.get()

    # Update password in v_user_info view
    update_sql = "UPDATE v_user_info SET password = %s WHERE username = %s"
    cursor.execute(update_sql, (new_password, username))

    db.commit()

    messagebox.showinfo("Success", "Password reset successfully!")

    # Check if forgot_window exists before destroying
    if forgot_window:
        forgot_window.destroy()
```

- **Deletion page**

The `update_window` is created with labels and entry fields for each column specified in the `columns` list. The user can input new values for the selected record, and upon clicking the "Update Record" button, the `perform_update` method is triggered. This method constructs and executes an SQL UPDATE query, using the provided table name (`table`), primary key (`key`), record ID (`id`), and the new values from the entry fields (`fields`). The `mysql.connector` library is used for database connectivity. Success or failure messages are displayed using the Tkinter `messagebox` module. The code concludes with an instantiation of the `AdminLoginWindow` class and the main loop for the Tkinter GUI. However, there is a small typo in the last part of the code; it should be `__name__` instead of `_name_`, and `__main__` instead of `_main_`.

CODE:

```
        update_fields = []
        for c in columns:
            field_label = ttk.Label(update_window, text=f"{c}:")
            field_label.pack()

            field = ttk.Entry(update_window)
            field.insert(tk.END, result[i])
            field.pack()

            update_fields.append(field)

        # Update button
        update_btn = ttk.Button(update_window, text="Update Record",
                                command=lambda: self.perform_update(table, primary_key, selected_record[0],
update_fields))
        update_btn.pack(pady=10)

    def perform_update(self, table, key, id, fields):
        try:
            set_cols = ",".join([f"{c}=%s" for c in columns])
            update_query = f"UPDATE {table} SET {set_cols} WHERE {key}=%s"

            # Execute the update query
            values = [field.get() for field in fields] + [id]
            self.cursor.execute(update_query, values)
            self.db.commit()

            messagebox.showinfo("Success", "Record updated successfully")

        except mysql.connector.Error as err:
            print(f"Database Error: {err}")
            messagebox.showerror("Error", "Failed to update record")

if _name_ == "_main_":
    admin_login_window = AdminLoginWindow()
    admin_login_window.mainloop()
```

- **Print all data**

The provided code defines a Python function named `search` that performs a search operation in a bus schedule database. The function utilizes global variables for user input fields such as starting location (`start_entry`), ending location (`end_entry`), day of travel (`day_entry`), and arrival time (`arrival_time_entry`). The user's input is retrieved, and an arrival time is converted into a timedelta object. The code then connects to a database, executes a SELECT query on a view named "BusScheduleView," and fetches the results. These results are filtered based on the user's specified arrival time. The code clears and updates a text widget (`result_text_widget`) to display the filtered bus schedule information. Additionally, for each result, a button labeled "Select and proceed" is created within the text widget, linking to a function named `payment_interface` with the selected result as a parameter. The error handling is included for potential database errors. Overall, this code snippet appears to be a part of a larger program that integrates a graphical user interface (GUI) with database connectivity to facilitate searching and displaying bus schedule information.

CODE:

```
def search():
    global start_entry, end_entry, day_entry, arrival_time_entry, result_text_widget  # Declare as
global variables

    # Get user input
    start = start_entry.get()
    end = end_entry.get()
    day = day_entry.get()
    arrival_time_str = arrival_time_entry.get()

    # Convert the user input arrival time to a timedelta object
    arrival_time = timedelta(hours=int(arrival_time_str.split(':')[0]),
minutes=int(arrival_time_str.split(':')[1]))

    # Database connection and search functionality
    try:
        # Construct and execute the database query
        cursor.execute("SELECT * FROM BusScheduleView")
        results = cursor.fetchall()

        # Filter results based on arrival time
        filtered_results = [result for result in results if result[1] >= arrival_time]

        # Clear previous search results in the scrolled text widget
        result_text_widget.delete(1.0, tk.END)

        # Display filtered results in the scrolled text widget
        for result in filtered_results:
            result_text = f"\n\nBus Number: {result[0]}\n"
            result_text += f"Arrival Time: {result[1]}\n"
            result_text += f"Departure Time: {result[2]}\n"
            result_text += f"Number of Stops: {result[3]}\n"
            result_text += "--------------------------\n"

            # Insert the result into the scrolled text widget
            result_text_widget.insert(tk.END, result_text)

            # Create a button for each result inside the scrolled text widget
            result_btn = tk.Button(result_text_widget, text="Select and proceed", command=lambda
result=result: payment_interface(result))
            result_text_widget.window_create(tk.END, window=result_btn)


    except mysql.connector.Error as err:
        print(f"Database Error: {err}")
```

12.REFERENCES

1. Dutchess County Public Transportation Application: https://www.dutchessny.gov/Departments/Public-Transit/Public-Transit.htm

2. NYC Transit: MTA Subway and Bus: https://transitapp.com/region/new-york
3. Moovit: https://moovitapp.com/nycnj-121/poi/en