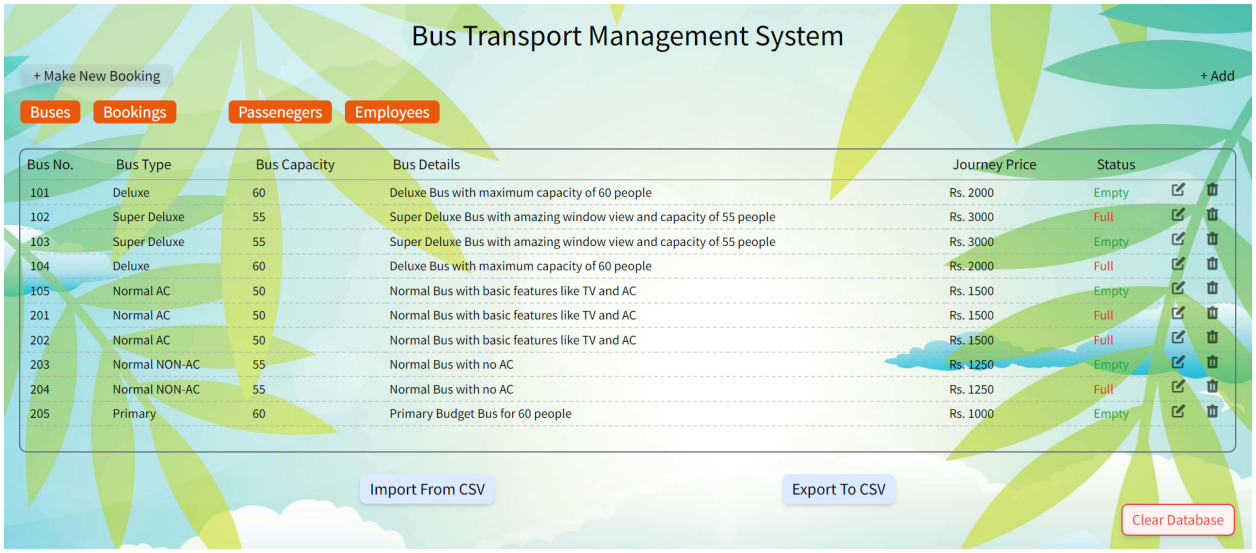


Transport Management System

Anshika 2021CSB1069

Darakshinda 2021CSB1130

Chaitanya 2021CSB1121

B.Tech 3rd Year

Department of Computer Science and Engineering, IIT Ropar

Github Link

Overview

The Transport Management System project simplifies and streamlines bus and taxi transportation services through the integration of a robust Database Management System (DBMS). The project leverages SQL for database management and Flask and Python for scripting and backend logic. The system focuses on enhancing efficiency, transparency, and user experience in managing transportation services.

Features

1. **Add and manage passenger details.**
2. **Add and manage employee details.**
3. **Make bus reservations and track transactions.**
4. **Import and export data to/from CSV files.**
5. **Clear all data in the database.**

Goals

6. **Automation of Operations:** Develop a comprehensive TMS Transport Management Systemto automate various aspects of transportation management, including route planning, scheduling, ticketing, and reporting.
7. **Database Management**: Implement a reliable and scalable database using SQL to store and manage data related to vehicles, routes, schedules, passengers, and transactions.
8. **User-Friendly Interface**: Create an intuitive and user-friendly interface for both administrators and end-users to facilitate easy interaction with the system.

Specifications

The proposed system utilizes a three-tier architecture:

1. **Presentation Layer:** Developed using Python-based frameworks (e.g., Flask) for a dynamic and responsive user interface.
2. **Application Layer**: Python scripts for backend logic, business rules, and seamless integration with the database.

3. **Data Layer**: A MySQL database managed using SQL to store and retrieve information related to buses, passengers, bus stops and transactions.

Contributions

- ### Database Structure and Queries in MySQL

  Schema Tables Design: Anshika, Darakshinda,Chaitanya

  Schema Tables Creation Queries: Anshika, Darakshinda

  Schema Data Manipulation Queries: Anshika, Darakshinda, Chaitanya

- ### Backend using Flask

  Backend design: Anshika, Darakshinda,Chaitanya

  Authorization: Anshika

  Backend Functions: Anshika, Darakshinda

- ### Frontend

  Frontend Design: Darakshinda,Chaitanya

  Frontend HTML: Anshika

- ### Report and ER Diagram

  Anshika, Darakshinda

Normalization Analysis

- ### First Normal Form

  A. **Atomic values:** Each table column has atomic (indivisible) values. The Passenger table's passen_fname and passen_lname columns include first and last names.

  B. **No Group Repetition:** No column has recurring groups or arrays. Each table cell has an indivisible value.

  C. **Unique Column Names**: Table columns have distinct names. No Passenger or other table has two columns with the same name.

- ### Second Normal Form

A. **Meets 1NF**: As noted, the schema meets first normal form criteria. Atomic values, no repeated groupings, and unique column names are in each column.

B. **No Partial Dependencies**: 2NF ensures that all non-prime characteristics (attributes not in any candidate key) are functionally reliant on the primary key, eliminating partial dependencies. Passenger table main key is passen_id.

   The main key controls all other properties (passen_fname, lname, address, ph_no, status).Res_id is the Reservation table's main key.

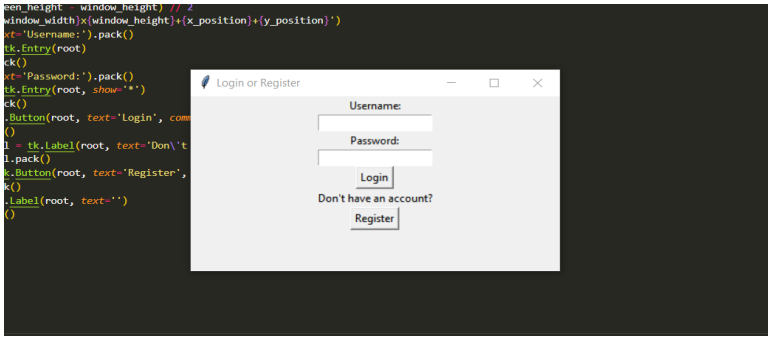
   No partial dependencies exist between the main key and the attributes (passen_id, bus_id, transaction_id, stop1_expected, stop1_real, stop2_expected, stop2_real, hours).Other tables like Bus, Bus_Type, Transactions, Employees, Job, and User have characteristics that are functionally reliant on their main keys.

C. **Proper Use of Foreign Keys**: Tables are linked via foreign keys. The Employees table's job_id foreign key references the Job table. This links the Employees and Job tables using the Job table's main key..

Special Features of the database:

1. Authorization:

Our database management system includes an authorization component where only the registered users can access the database. First you have to run the app.py then a GUI appears to ask you for your username and password. If you are already registered, you can simply enter your details and access the database, else you will have to first register. This authorization uses a database for its working using sqlite3.



2. Functionality:

The database incorporates a special functionality that uses the generation of lateness fine. This lateness fine is calculated for every 15 minute lateness at a rate of Rs 5 per 15 minute delay.

Further the lateness is reflected in the bus status as '**Delayed**' if its late and '**On time**' if its on time .

```
CREATE TABLE Reservation (
    res_id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
    passen_id INT,
    bus_id INT,
    transaction_id VARCHAR(12),
    stop1_expected DATETIME,
    stop1_real DATETIME,
    stop2_expected DATETIME,
    stop2_real DATETIME,
    stop3_expected DATETIME,
    stop3_real DATETIME,
    lateness_fine REAL GENERATED ALWAYS AS (
        GREATEST(0,
            (CASE WHEN TIMESTAMPDIFF(MINUTE, stop1_expected, stop1_real) > 15 THEN (TIMESTAMPDIFF(MINUTE, stop1_expected, stop1_real) - 15) * 5 ELSE 0 END) +
            (CASE WHEN TIMESTAMPDIFF(MINUTE, stop2_expected, stop2_real) > 15 THEN (TIMESTAMPDIFF(MINUTE, stop2_expected, stop2_real) - 15) * 5 ELSE 0 END) +
            (CASE WHEN TIMESTAMPDIFF(MINUTE, stop3_expected, stop3_real) > 15 THEN (TIMESTAMPDIFF(MINUTE, stop3_expected, stop3_real) - 15) * 5 ELSE 0 END)
        )
    ) STORED,
    status VARCHAR(50)
);
```

3. Triggers:

Our database management system includes two special triggers (refer triggers.sql).

1. The first trigger is to **set the default status (delayed or on time)** of the bus based on initial expected and real time fed into the database.
2. The second is the most important trigger that sets the **lateness fine** based on the latest real time to reach a atop entered to the bus. An example is also provided in triggers.sql to check the working of the same.

```
---trigger 1 to update status on time orr delay
DELIMITER //
CREATE TRIGGER SetDefaultStatus
BEFORE INSERT ON Reservation
FOR EACH ROW
BEGIN
    SET NEW.status = CASE WHEN NEW.lateness_fine > 0 THEN 'Delayed' ELSE 'On Time' END;
END;
//
DELIMITER ;
--trigger 2 to update



CREATE TRIGGER UpdateReservationStatus
BEFORE UPDATE ON Reservation
FOR EACH ROW
BEGIN
    SET NEW.stop3_real = CASE WHEN TIMESTAMPDIFF(MINUTE, NEW.stop3_expected, NEW.stop3_real) > 15 THEN NEW.stop3_real ELSE NEW.stop3_expected + INTERVAL 15 MINUTE END;

    SET NEW.lateness_fine = GREATEST(0,
        (CASE WHEN TIMESTAMPDIFF(MINUTE, NEW.stop1_expected, NEW.stop1_real) > 15 THEN (TIMESTAMPDIFF(MINUTE, NEW.stop1_expected, NEW.stop1_real) - 15) * 5 ELSE 0 END) +
        (CASE WHEN TIMESTAMPDIFF(MINUTE, NEW.stop2_expected, NEW.stop2_real) > 15 THEN (TIMESTAMPDIFF(MINUTE, NEW.stop2_expected, NEW.stop2_real) - 15) * 5 ELSE 0 END) +
        (CASE WHEN TIMESTAMPDIFF(MINUTE, NEW.stop3_expected, NEW.stop3_real) > 15 THEN (TIMESTAMPDIFF(MINUTE, NEW.stop3_expected, NEW.stop3_real) - 15) * 5 ELSE 0 END)
    );

    SET NEW.status = CASE WHEN NEW.lateness_fine > 0 THEN 'Delayed' ELSE 'On Time' END;
END;
//
DELIMITER ;
```

4. Functions and Procedures:

Our database is equipped with several functionalities that includes a function for **calculating the total amount spent by any passengers on all the bookings till date** and a procedure to **calculate the total number of minutes that a bus delayed in a journey.**

For these refer file functions.sql

```
CREATE FUNCTION CalculateTotalAmount(passenID INT)
RETURNS INT
BEGIN
    DECLARE totalAmount INT;
    SELECT SUM(amount) INTO totalAmount
    FROM Transactions
    WHERE res_id IN (SELECT res_id FROM Reservation WHERE passen_id = passenID);
    RETURN totalAmount;
END;
//
DELIMITER ;

SELECT CalculateTotalAmount(1) AS total_amount_for_passenger_1;


DELIMITER //
CREATE PROCEDURE CalculateTotalDelay(IN busID INT, OUT totalDelayMinutes INT)
BEGIN
    DECLARE totalDelay INT DEFAULT 0;

    -- Calculate delay for each reservation and accumulate
    SELECT
        SUM(
            GREATEST(0,
                (CASE WHEN TIMESTAMPDIFF(MINUTE, stop1_expected, stop1_real) > 15 THEN (TIMESTAMPDIFF(MINUTE, stop1_expected, stop1_real) - 15) ELSE 0 END) +
                (CASE WHEN TIMESTAMPDIFF(MINUTE, stop2_expected, stop2_real) > 15 THEN (TIMESTAMPDIFF(MINUTE, stop2_expected, stop2_real) - 15) ELSE 0 END) +
                (CASE WHEN TIMESTAMPDIFF(MINUTE, stop3_expected, stop3_real) > 15 THEN (TIMESTAMPDIFF(MINUTE, stop3_expected, stop3_real) - 15) ELSE 0 END)
            )
        ) INTO totalDelay
    FROM Reservation
    WHERE bus_id = busID;
```

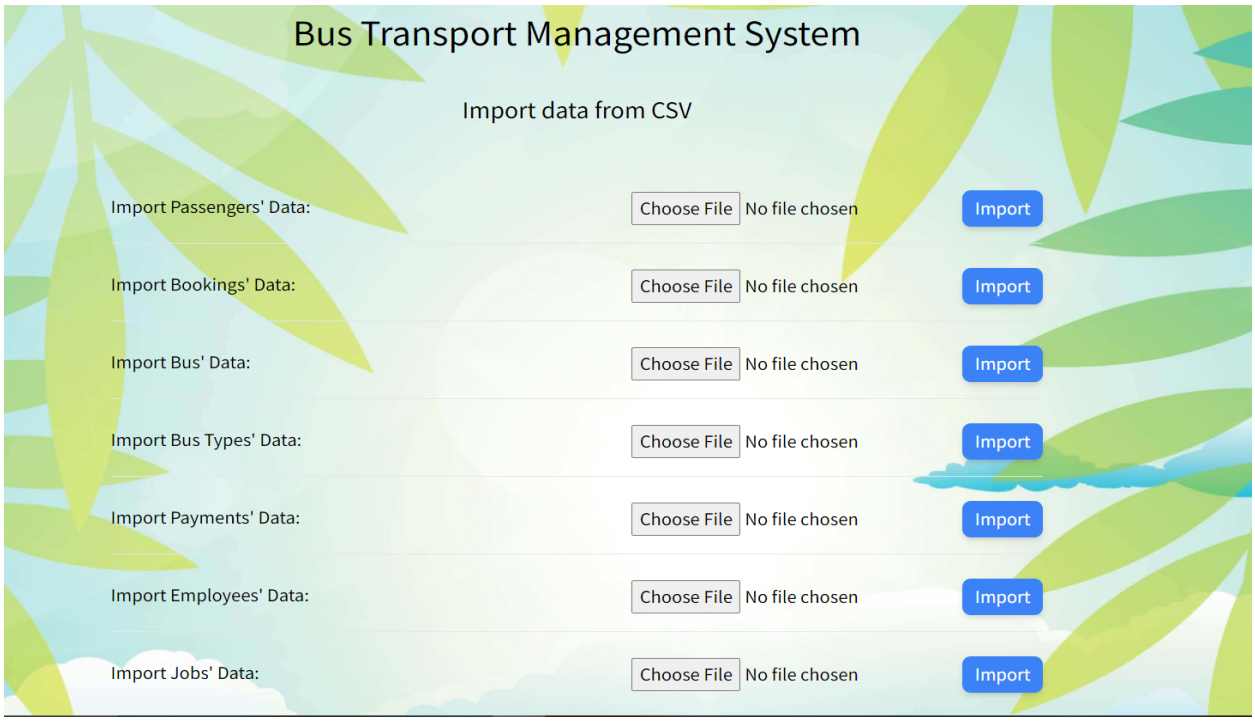
5. Views:

Our database supports several views(refer views.sql) that inlcude:

1. View to Show Reservations with Passenger Information
2. View to Show Bus Information with Type
3. View to Show Employee Information with Job Title

6. Special feature:

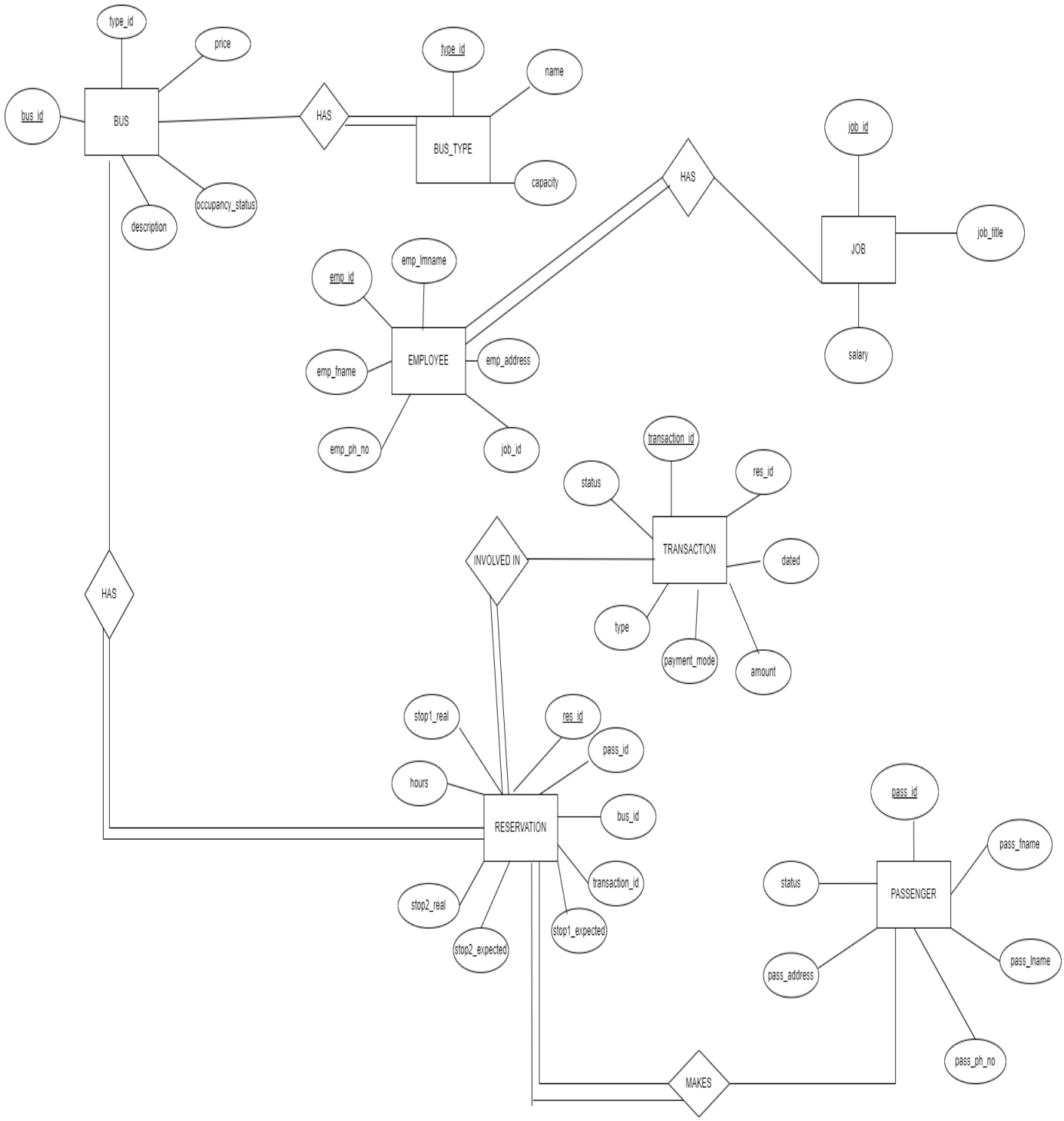
The special feature of our database is that one can import the data into tables using CSV files instead of having to write all the queries. Also the data from tables can be exported as CSV files from the database itself.



How to Run:

1. Run app.y
2. Register into the database from the GUI that pops up on running app.py
3. Login to the database
4. In Mysql ‘newdb’ database will be created
5. Open ordered.sql and copy its queries. Run them in Mysql
6. In the command line of the ide where you run app.py, there will be a link for localhost. Open that link and see the database having entries as required.

ER Diagram:


type_id
price
bus_id
BUS
HAS
type_id
name
BUS_TYPE
capacity
occupancy_status
description
job_id
HAS
job_title
JOB
salary
emp_id
emp_lmname
emp_fname
EMPLOYEE
emp_address
emp_ph_no
job_id
transaction_id
status
res_id
INVOLVED IN
TRANSACTION
dated
type
payment_mode
amount
HAS
stop1_real
res_id
pass_id
hours
RESERVATION
bus_id
transaction_id
stop2_real
stop1_expected
stop2_expected
pass_id
pass_fname
status
PASSENGER
pass_lname
pass_address
pass_ph_no
MAKES

Cardinality:

1. Many **Passengers** can have Many **Reservation**s (M:M)
2. One **Bus** can have Many **Reservations** (1:M)
3. One **Bus Type** can have Many **Buse**s (1:M)
4. One **transaction** can have only one **reservation** (1:1)
5. One **Job** can have Many **Employees** (1:M)