جامعة مصر للمعلوماتية
EGYPT UNIVERSITY OF INFORMATICS

FACULTY OF ENGINEERING

Railway Booking System

Team Members:

AbdelRahman Safwat	21-101108
Hamdi Awad	21-101011
Omar Walid	21-101031
Doha Bahaaeldin	21-101136
Kareem AbdelHameed	21-101015

Course Instructors:

Prof. Hala Zayed

Eng. Rameez Barakat

Course:

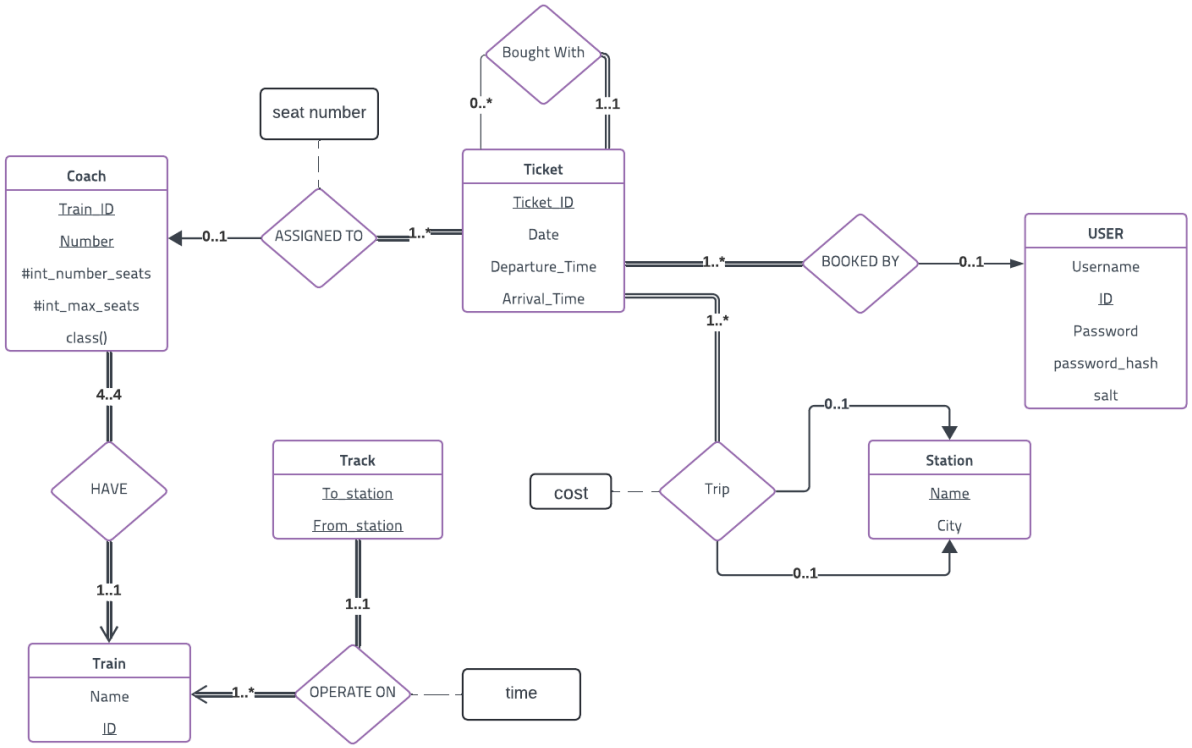
Database Systems (CSE 371)

جامعة مصر للمعلوماتية
EGYPT UNIVERSITY OF INFORMATICS

FACULTY OF ENGINEERING

Introduction:

This project aims to design and implement a comprehensive railway system database. The database is structured to manage various aspects of a railway system, including stations, tracks, trains, coaches, tickets, and users. The goal is to facilitate efficient operations and ensure seamless interactions between different entities within the railway system. Below is the ER diagram for our project:



Database Schema

The database schema includes the following relations:

1. **ASSIGNED TO:**

   Describes the relation between ticket and coach where one ticket or more can be assigned to one coach but only one coach is assigned to each ticket.

   - Attributes:

     seat number

2. **Trip:**
   A ticket may contain one `From_station` or one `To_station` or both. However, a ticket has to contain at least one station and can contain more than two stations.

   - Attributes:

     cost

3. **HAVE:**

   Each train must have four coaches and only four coaches, and each coach belongs to exactly one train.

4. **OPERATE ON:**

   A train can have only one track, no more or less. A track can be operated on by many trains but at least one train has to operate on each track.

   - Attributes:

     Time

5. **BOOKED BY:**

   A user can have one or more than one ticket. A ticket can only be owned by a maximum of one user or not owned by any user.

6. **Bought With:**

   Multiple Tickets can be bought together under the same ticket id so that deletion and modification can be applied on the tickets together.

Note the following:

All the relations we used are redundant which is why we added all primary attributes of the one side to the many side .
For example, the Assigned to relation between the ticket and coach is redundant so this means that the prime attributes in the one sided (coach) will be foreign attributes in the many sided (ticket).

Below you can find all SQL tables with their details:

Ticket Table:

1. **Ticket:**

- Attributes:

`Ticket_ID:` The unique identifier for the ticket.
`Together_ID:` The multiple ticket identifier
`Cost:` The cost of the ticket.
`Class:` The class of the ticket.
`Date:` The date of travel.
`Departure_Time:` The departure time.
`Arrival_Time:` The arrival time.
`From_station:` The starting station of the track.
`To_station:` The ending station of the track.

`Coach_Number:` The number of the coach.
`Seat_no:` The seat number assigned to the passenger.
`Username:` The name of the passenger.

Column Name	Data Type	Required Value	Key	Default Values	Remarks
Ticket_ID	INT	NO	Primary Key	NULL	AUTO INCREMENT
Together_ID	INT	YES	NONE	NULL	NONE
Train_ID	NUMERIC(10)	YES	Foreign Key	NULL	REF: Coach(Train_ID)
Departure_time	TIME	YES	NONE	NULL	NONE
Arrival_time	TIME	YES	NONE	NULL	NONE
From_Station	VARCHAR(60)	YES	Foreign Key	NULL	REF: Station(Name)
To_Station	VARCHAR(60)	YES	Foreign Key	NULL	REF: Station(Name)
Coach_Number	NUMERIC(1)	YES	Foreign Key	NULL	REF: Coach(Coach_Number)
Seat_no	NUMERIC(3)	YES	NONE	NULL	NONE
Username	VARCHAR(50)	YES	Foreign Key	NULL	REF: users(Username)

Station Table:

2. **Station:**

- Attributes:

`Name:` The name of the station.
`City:` The City where the station is located.

Column Name	Data Type	Required Value	Key	Default Values	Remarks
Name	VARCHAR(60)	NO	Primary Key	NULL	NONE
City	VARCHAR(20)	YES	NONE	NULL	NONE

User Table:

3. **USER:**

- Attributes:

`ID:` The unique identifier for the user.
`Username:` The username of the user.
`Password_hash:` The hashed version of the user's password.
`Salt:` The salt used in hashing the password.

Column Name	Data Type	Required Value	Key	Default Values	Remarks
ID	INT	NO	Primary Key	NULL	AUTO INCREMENT
username	VARCHAR(50)	NO	Unique Key	NULL	NONE
password_hash	VARCHAR(255)	NO	NONE	NULL	NONE
Salt	VARCHAR(255)	YES	NONE	NULL	NONE

Track Table:

4. **Track:**

- Attributes:

`From_station:` The starting station of the track.
`To_station:` The ending station of the track.
`Train_ID:` The unique identifier for the train.
`Dept_time:` The time that the train departs.

Column Name	Data Type	Required Value	Key	Default Values	Remarks
From_station	VARCHAR (60)	NO	Primary, Foreign	NULL	REF: Station(Name)
To_station	VARCHAR (60)	NO	Primary, Foreign	NULL	REF: Station(Name)
Train_ID	NUMERIC(10)	YES	Primary, Foreign	NULL	REF: Train(Train_ID)
Dept_time	NUMERIC(2)	YES		NULL	NONE

Coach Table:

5. **Coach:**

- Attributes:

`Train_ID:` The ID of the train to which the coach is assigned.

`Coach_Number:` The number of the coach.
`Seats_array:` The information of the seat.
`Max_seats:` The maximum capacity of the coach.

Column Name	Data Type	Required Value	Key	Default Values	Remarks
Train_ID	NUMERIC(10)	NO	Primary, Foreign	NULL	REF: Train(Train_ID)
Coach_number	NUMERIC(1)	NO	Primary Key	NULL	NONE
Seats_array	NUMERIC(8)	YES	NONE	NULL	NONE
Max_seats	NUMERIC(8)	YES	NONE	NULL	NONE

Train table:

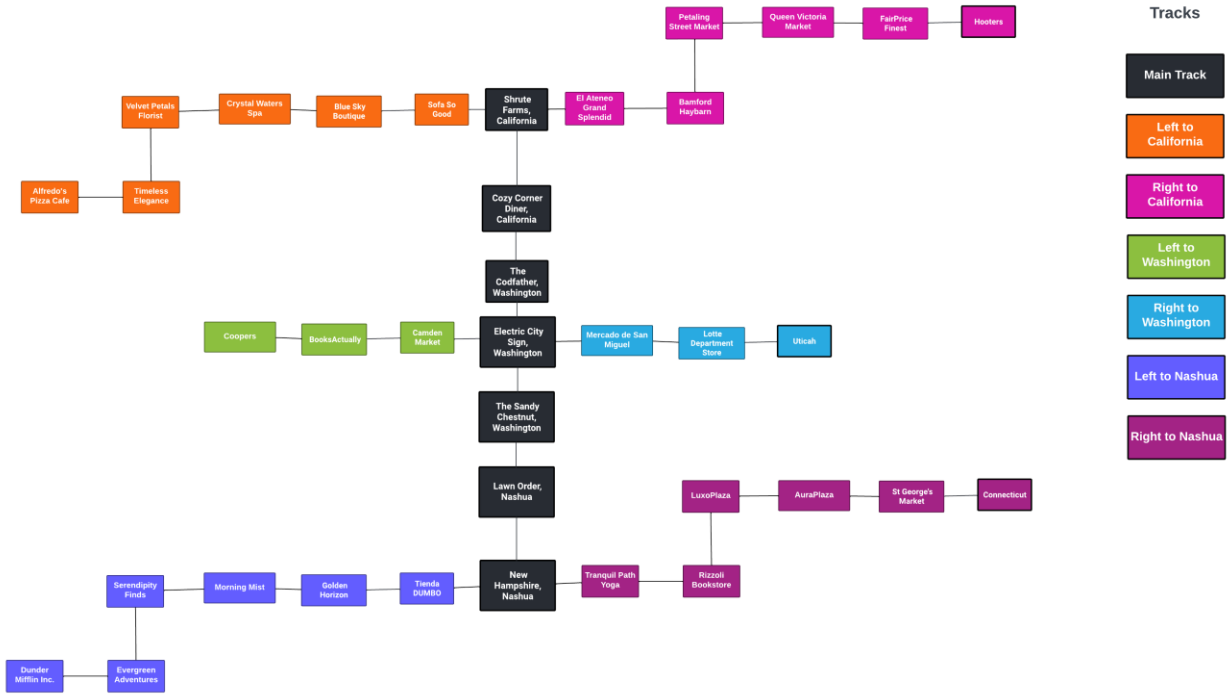
6. **Train:**

- Attributes:

`Name:` The name of the train.
`ID:` The unique identifier for the train.

Column Name	Data Type	Required Value	Key	Default Values	Remarks
Name	VARCHAR(20)	YES	NONE	NULL	NONE
Train_ID	NUMERIC(10)	NO	Primary Key	NULL	NONE

Stations Map:



In our model, we assume that between each station and the adjacent one an hour-long trip. Stations located on the main track can also be transitional stations where one can transition their track to head to their desired station. Different colors can demonstrate each track and the concept of transitions.

FrontEnd/BackEnd:

MYSQL CONNECTOR:

The MySQL connector is vital as it sets up a connection to the MySQL database and returns a cursor for executing SQL queries and managing transactions. It ensures that all database interactions are properly managed and encapsulated, providing a foundation for all subsequent operations.

CreateConnection

- **create()**: Establishes a connection to the MySQL database.
  CreateDatabase
- **check_and_create_table(backEnd, table_name, create_table_sql):** Checks for the existence of a table and creates it if not found.
- **Construct_Database():** Constructs the entire database schema if it doesn't exist.
  GUI_Authentication
- **login(app):** Handles user login.
- **update_sidebar_after_login(app, username):** Updates the sidebar after a successful login.
- **logout(app):** Handles user logout.
- **register(app):** Handles user registration.
  GUI_booking
- **booking_page(app):** Displays the booking page.
- **update_available_trains(app, event):** Updates the list of available trains based on the selected route.
- **get_available_trains(from_location, to_location):** Retrieves available trains for the specified route.
- **book_tickets(app):** Handles the ticket booking process.
- **clear_booking_form(app):** Clears the booking form.
  GUI_UI
- **create_login_form_ui(app):** Creates the login form UI.
- **update_appearance_mode(app):** Updates the app's appearance mode.
- **clear_sidebar(app):** Clears the sidebar.
- **clear_main_content(app):** Clears the main content area.
  InsertData
- This file specifies in inserting the values of each table and linking them with each other for example **create_train(Name, Train_ID)** Inserts a new train into the database. And **create_Coach(Train_ID, Coach_Number, Seats_array, Max_Seats)** Inserts a new coach, the same with **create_stations(stations)** which Inserts

multiple stations into the database, and **create_track(from_station, to_station, train_id, time)** which follow the same pattern and concept, I think you get the idea

loginVerifier

- **create_user(username, hashed_password, salt):** Creates a new user.
- **hash_password(password):** Hashes a password.
- **verify_password(password, hashed_password, salt):** Verifies a password against a stored hash.

Use Case Scenarios:

★ **Booking a Ticket:**

A user logs into the system, selects the desired departure and arrival stations, and books a ticket for a specific train and class. The system generates a ticket with a unique ID, assigns a seat, and updates the database accordingly.

★ **Managing Trains and Coaches:**

The railway administration can add new trains and coaches to the system, assign coaches to trains, and update the seating capacities and classes of the coaches.

★ **Scheduling and Tracking:**

The system manages the scheduling of trains on various tracks and keeps track of the departure and arrival times at different stations.

Goals:

★ **Efficiency:** Ensure the system can handle large volumes of data and transactions efficiently.

- **Scalability:** Design the database to accommodate future expansions, such as additional stations, tracks, and trains.
- **Security:** Implement robust security measures like encrypting to protect user data, especially for user authentication and ticket bookings.

By achieving these goals, the railway system database will facilitate smooth and efficient railway operations, enhance user experience, and provide a reliable platform for managing railway resources.

Conclusion:

The Railway Booking System project represents a comprehensive approach to managing the various components of a railway network, ensuring efficient operations and seamless interactions between stations, tracks, trains, coaches, tickets, and users. By designing and implementing a well-structured database schema, we have established a robust foundation that supports key functionalities such as booking tickets, managing trains and coaches, and scheduling and tracking train operations.

Our database system leverages relational integrity and structured query language (SQL) to ensure that data is accurately captured, stored, and maintained. The inclusion of foreign keys, primary keys, and necessary attributes across multiple tables allows for a cohesive and interconnected data model. This ensures that all entities within the railway system can be managed and queried efficiently.