Train stations system (TSS)

Student (1) name: Ahmed Mohamed Abdel Hamid

Student (1) ID: 221011073

Student (2) name: Omar Khaled

Student (2) ID: 221002374

College: Artificial Intelligence (AI)

Course instructor: Dr. Omar Shalash

TA: Eng. Salma Mohammed

Table of Contents

1. Introduction

The train stations system **(TSS)** represents a sophisticated and meticulously designed database solution poised to revolutionize the operational landscape of train stations across diverse regions. Recognizing the intricacies inherent in managing multiple stations, the **TSS** is tailored to optimize and streamline the multifaceted operations within this sector. This comprehensive system serves as the backbone for orchestrating seamless interactions among stations, platforms, employees, schedules, tickets, trains, reviews, passengers, payments, bookings, and transactions.

Strategic Management of Station Networks:

At its core, the **TSS** is engineered to address the complexities of managing not just individual stations but entire networks. The database captures crucial information about each station, including its unique identifier, name, capacity, government affiliation, region, and physical address. By establishing hierarchical relationships, the system seamlessly navigates through station structures, enabling efficient management and connectivity across a network of train stations.

Platform Utilization and Coordination:

Efficient platform utilization is a focal point of the **TSS**, with the "Platform" module intricately tracking platform numbers, associated stations, and schedules. Foreign key constraints are strategically implemented to ensure data integrity, facilitating smooth coordination of platform-related activities across multiple stations simultaneously.

Comprehensive Employee Deployment:

The **TSS** acknowledges the significance of a well-organized and dynamic workforce, and its "Employee" module systematically manages employee data. This includes details such as names, salary, positions, and station affiliations, establishing a transparent organizational hierarchy across all stations for effective workforce deployment based on demand and station capacity.

Robust Scheduling Mechanism:

A robust scheduling mechanism lies at the heart of the **TSS**, capturing critical details about schedules, states, and arrival times. This intricate web of information serves as the foundation for orchestrating seamless train arrivals and departures across multiple stations, ensuring optimal coordination and adherence to tight schedules.

Efficient Ticketing Operations:

The **TSS** ensures efficient ticketing operations through the "Ticket" module, managing ticket-related data such as price, trip names, and schedule IDs. This module establishes a pivotal link to the scheduling system, enabling harmonized coordination of passenger travel across diverse stations within the network.

Insights into Train Management:

Insights into train details, encompassing names, capacity, materials used, production years, and schedule IDs, are provided by the "Train" module. This information is central to effective train management, ensuring a harmonious distribution of resources and streamlined operations across the entire network.

Passenger-Centric Approach:

The **TSS** introduces a "Review" module to capture passenger feedback, contributing to a continuous improvement cycle. Ratings provided by passengers offer valuable insights into service quality and facilitate a passenger-centric approach across the diverse array of stations.

Transparent Financial Operations:

Financial transactions are meticulously tracked through the "Payment" module, capturing payment details such as amounts and registration dates. This level of transparency ensures accountability in financial operations, fostering reliability and confidence in the **TSS**.

2. Database Schema Overview

The Train Station System **(TSS)** database is a comprehensive and meticulously designed schema that captures the intricacies of managing a dynamic train station ecosystem. With entities ranging from fundamental station information to intricate financial transactions and passenger reviews, the schema provides a robust foundation for efficient and interconnected data management.

Station Entity:

Captures fundamental details about train stations, including station ID, name, capacity, government affiliation, region, street, and a reference to another station for hierarchical relationships.

Platform Entity:

Manages platform information with platform ID, number, and references to both stations and schedules. This allows for seamless coordination of platform activities and schedules.

Employee Entity:

Stores employee details, including employee ID, first name, last name, salary, position, date of birth, and a reference to the associated station. Enables transparent management of workforce deployment.

Emp_phone Entity:

Manages contact information for employees, providing a straightforward link between employee IDs and phone numbers.

Emp_age View:

A dynamic view calculating the age of employees based on their date of birth, providing insights into the workforce demographics.

Schedule Entity:

Captures schedule information with schedule ID, state, and arrival time, forming the foundation for managing train arrivals and departures.

Ticket Entity:

Manages ticket-related data, including ticket ID, price, trip name, and a reference to the associated schedule. Facilitates efficient ticketing operations.

Place Entity:

Establishes associations between stations and trains with station ID and train ID, allowing for structured organization and connectivity within the system.

Train Entity:

Stores comprehensive details about trains, including train ID, name, capacity, materials used, production year, and a reference to the associated schedule. Crucial for effective train management.

Review Entity:

Manages passenger reviews with review ID and rating information, facilitating continuous improvement in service quality.

Passenger Entity:

Stores passenger details, including passenger ID, first name, last name, and a reference to the associated review. Personalizes passenger experiences and feedback.

Pass_Contact Entity:

Manages contact details for passengers, including phone numbers and Gmail addresses, enhancing communication and service personalization.

Payment Entity:

Tracks financial transactions with payment ID, amount, and registration date, ensuring transparency and accountability in financial operations.

Booking Entity:

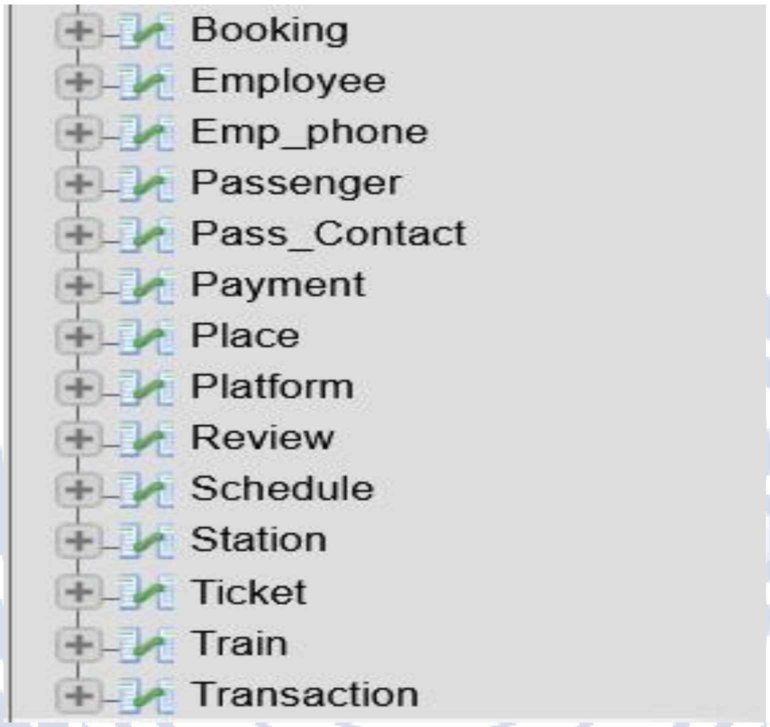
Manages booking information with booking ID, registration date, and a reference to the associated passenger. Essential for tracking and coordinating passenger travel plans.

Transaction Entity:

Represents the relationship between payments and bookings, ensuring that financial transactions align with the corresponding booking processes.

3. Entity Descriptions

This section provides details on each entity and their respective attributes within the Train Station System database. Each entity plays a crucial role in the overall database structure, and their attributes are carefully chosen to reflect the necessary data points for efficient **(TSS)** system.



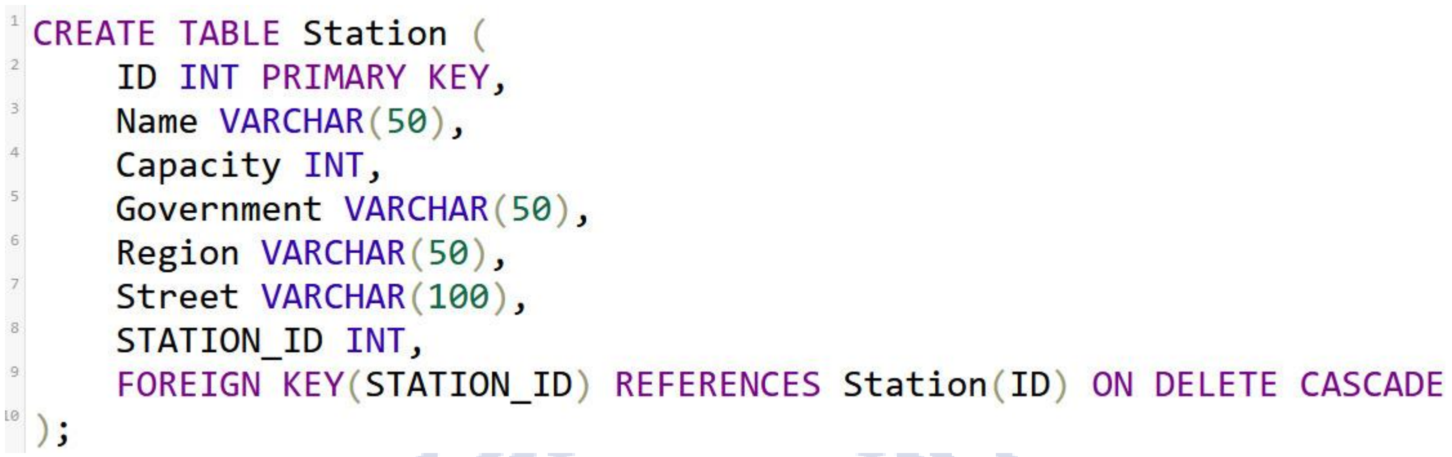
Booking
Employee
Emp_phone
Passenger
Pass_Contact
Payment
Place
Platform
Review
Schedule
Station
Ticket
Train
Transaction

3.1. Station

- **Attributes:**
  - ID: Unique identifier for the station.
  - Name: Name of the station.
  - Capacity: Maximum capacity of the station.
  - Government: Government affiliation of the station.
  - Region: Geographical region of the station.
  - Street: Physical street address of the station.
  - STATION_ID: Reference to another station for hierarchical relationships.
- **Description:**

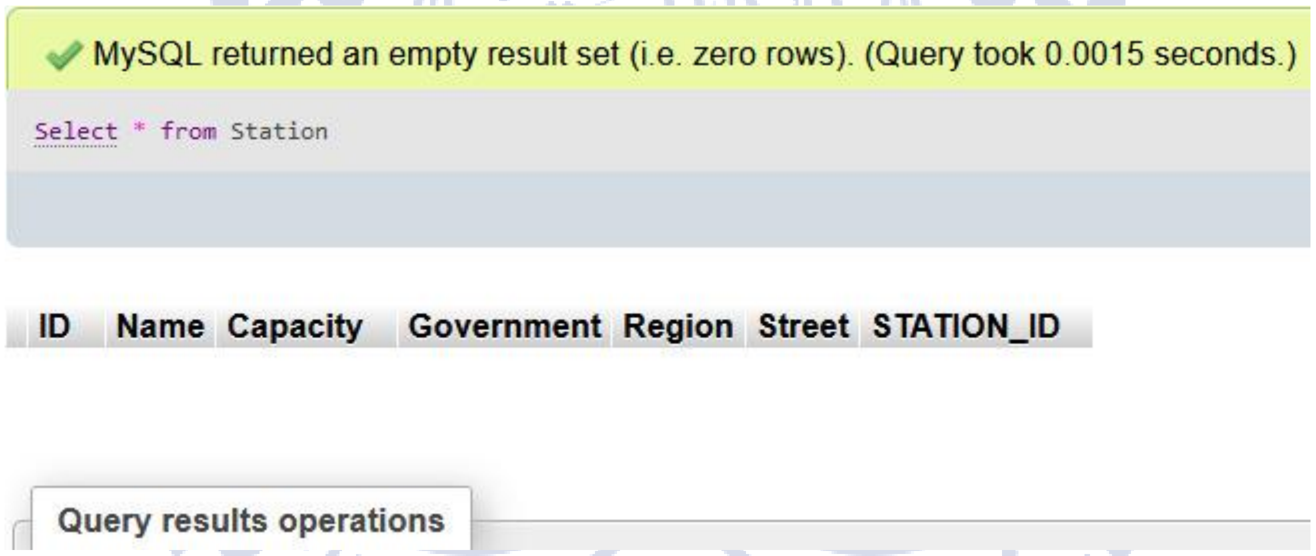
The Station entity represents the foundational information about each train station. It includes details such as station name, capacity, government affiliation, region, and physical address. The STATION_ID establishes hierarchical relationships among stations.

SQL Query:



```sql
CREATE TABLE Station (
    ID INT PRIMARY KEY,
    Name VARCHAR(50),
    Capacity INT,
    Government VARCHAR(50),
    Region VARCHAR(50),
    Street VARCHAR(100),
    STATION_ID INT,
    FOREIGN KEY(STATION_ID) REFERENCES Station(ID) ON DELETE CASCADE
);
```

Before Insertion:

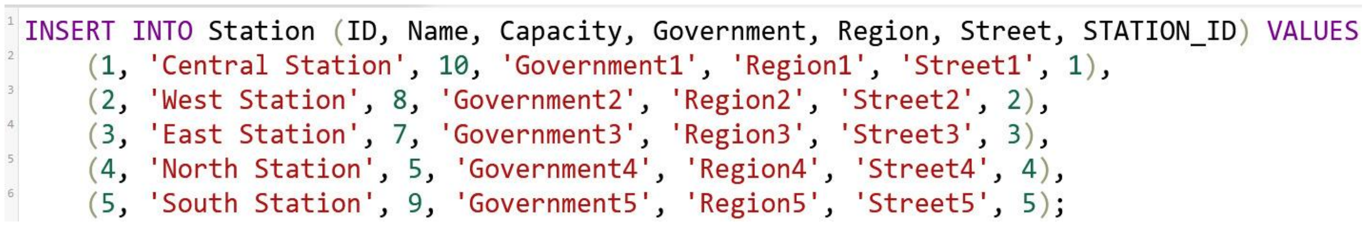


✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0015 seconds.)

```sql
Select * from Station
```

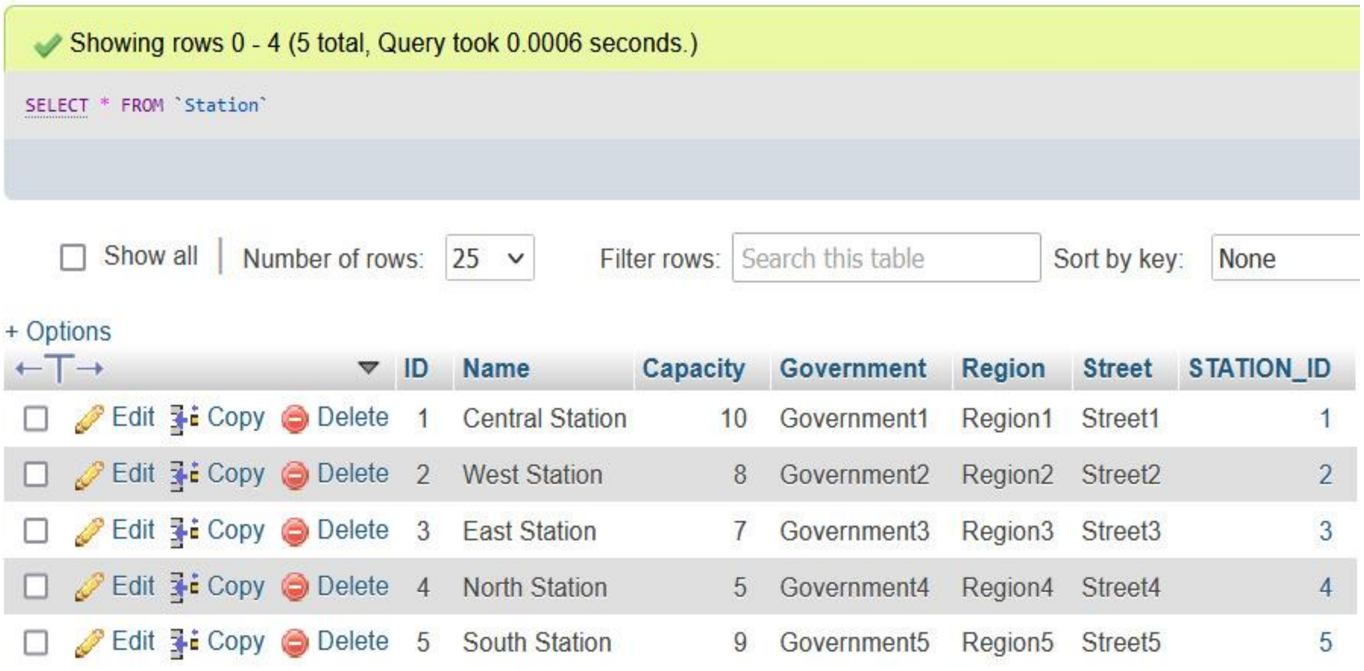
ID	Name	Capacity	Government	Region	Street	STATION_ID

Insertion:



```sql
INSERT INTO Station (ID, Name, Capacity, Government, Region, Street, STATION_ID) VALUES
    (1, 'Central Station', 10, 'Government1', 'Region1', 'Street1', 1),
    (2, 'West Station', 8, 'Government2', 'Region2', 'Street2', 2),
    (3, 'East Station', 7, 'Government3', 'Region3', 'Street3', 3),
    (4, 'North Station', 5, 'Government4', 'Region4', 'Street4', 4),
    (5, 'South Station', 9, 'Government5', 'Region5', 'Street5', 5);
```

After insertion:



✓ Showing rows 0 - 4 (5 total, Query took 0.0006 seconds.)

```
SELECT * FROM `Station`
```

	ID	Name	Capacity	Government	Region	Street	STATION_ID
Edit Copy Delete	1	Central Station	10	Government1	Region1	Street1	1
Edit Copy Delete	2	West Station	8	Government2	Region2	Street2	2
Edit Copy Delete	3	East Station	7	Government3	Region3	Street3	3
Edit Copy Delete	4	North Station	5	Government4	Region4	Street4	4
Edit Copy Delete	5	South Station	9	Government5	Region5	Street5	5

3.2. Employee

- **Attributes:**
  - ID: Unique identifier for the employee.
  - FName: First name of the employee.
  - Lname: Last name of the employee.
  - Salary: Salary of the employee.
  - Position: Job position of the employee.
  - DOB: Date of birth of the employee.
  - Station_ID: Reference to the associated station.

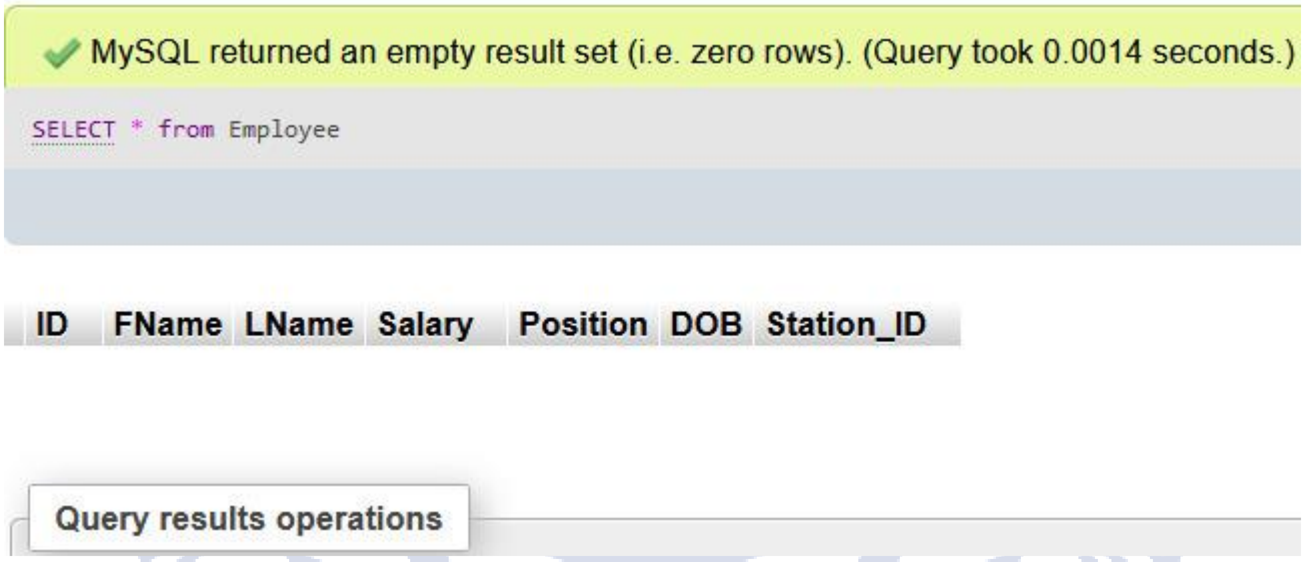
- **Description:**

The Employee entity stores details about each employee working within the train station system. It includes personal information, salary, position, date of birth, and a reference to the associated station for workforce management.

SQL Query:

```sql
CREATE TABLE Employee (
    ID INT PRIMARY KEY,
    FName VARCHAR(50),
    LName VARCHAR(50),
    Salary DECIMAL(10, 2),
    Position VARCHAR(50),
    DOB DATE,
    Station_ID INT,
    FOREIGN KEY (Station_ID) REFERENCES Station(ID) ON DELETE CASCADE
);
```

Before Insertion:


✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0014 seconds.)

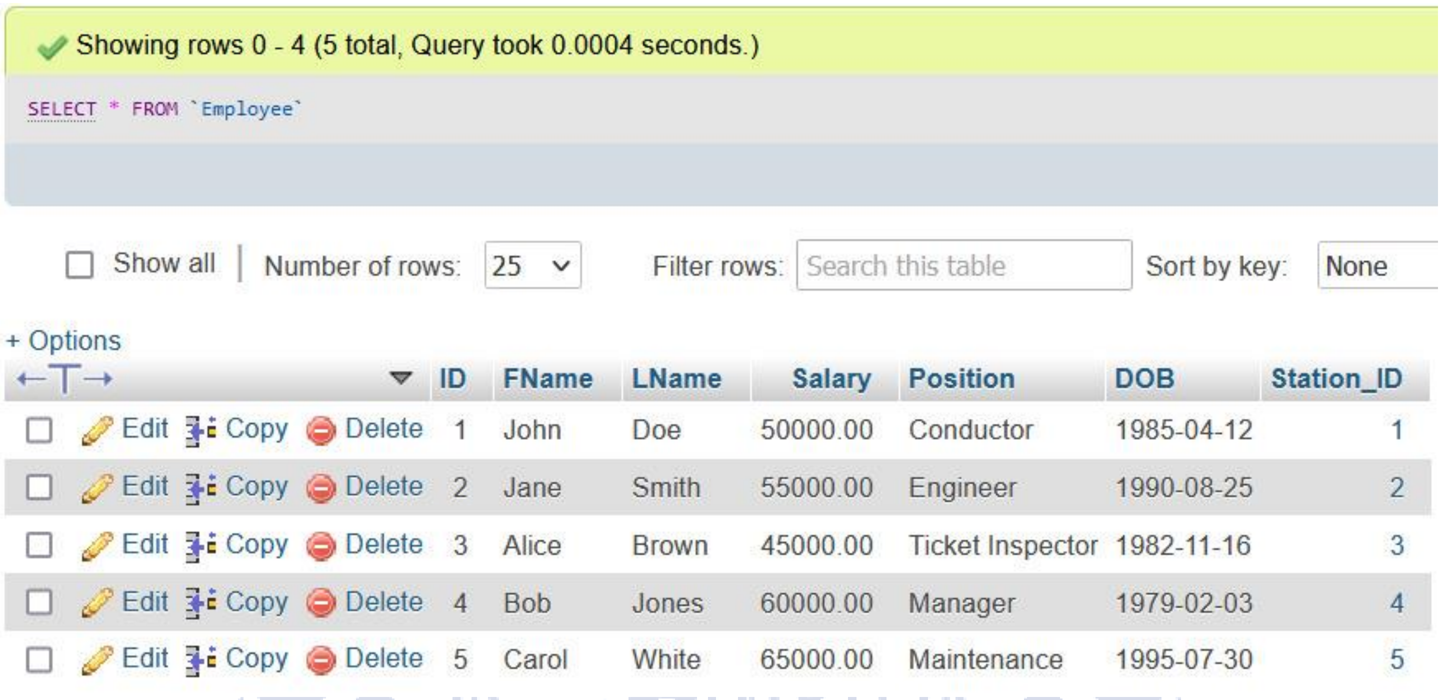
```sql
SELECT * from Employee
```

ID	FName	LName	Salary	Position	DOB	Station_ID

Insertion:

```sql
INSERT INTO Employee (ID, FName, LName, Salary, Position, DOB, Station_ID) VALUES
    (1, 'John', 'Doe', 50000, 'Conductor', '1985-04-12', 1),
    (2, 'Jane', 'Smith', 55000, 'Engineer', '1990-08-25', 2),
    (3, 'Alice', 'Brown', 45000, 'Ticket Inspector', '1982-11-16', 3),
    (4, 'Bob', 'Jones', 60000, 'Manager', '1979-02-03', 4),
    (5, 'Carol', 'White', 65000, 'Maintenance', '1995-07-30', 5);
```

After insertion:



Showing rows 0 - 4 (5 total, Query took 0.0004 seconds.)

```
SELECT * FROM `Employee`
```

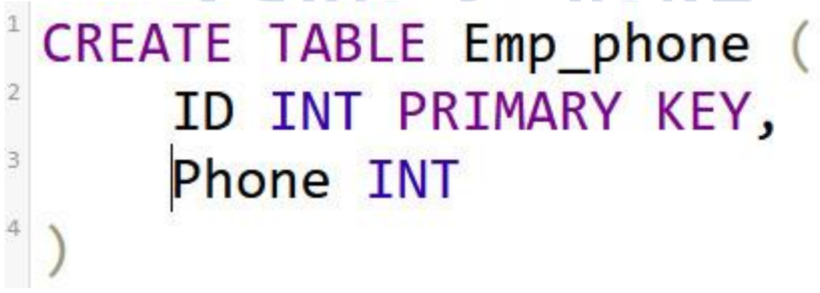
	ID	FName	LName	Salary	Position	DOB	Station_ID
Edit Copy Delete	1	John	Doe	50000.00	Conductor	1985-04-12	1
Edit Copy Delete	2	Jane	Smith	55000.00	Engineer	1990-08-25	2
Edit Copy Delete	3	Alice	Brown	45000.00	Ticket Inspector	1982-11-16	3
Edit Copy Delete	4	Bob	Jones	60000.00	Manager	1979-02-03	4
Edit Copy Delete	5	Carol	White	65000.00	Maintenance	1995-07-30	5

& Emp_phone (multi value in employee)

- **Attributes:**
  - ID: Unique identifier for employee phone information.
  - Phone: Phone number of the employee.
- **Description:**

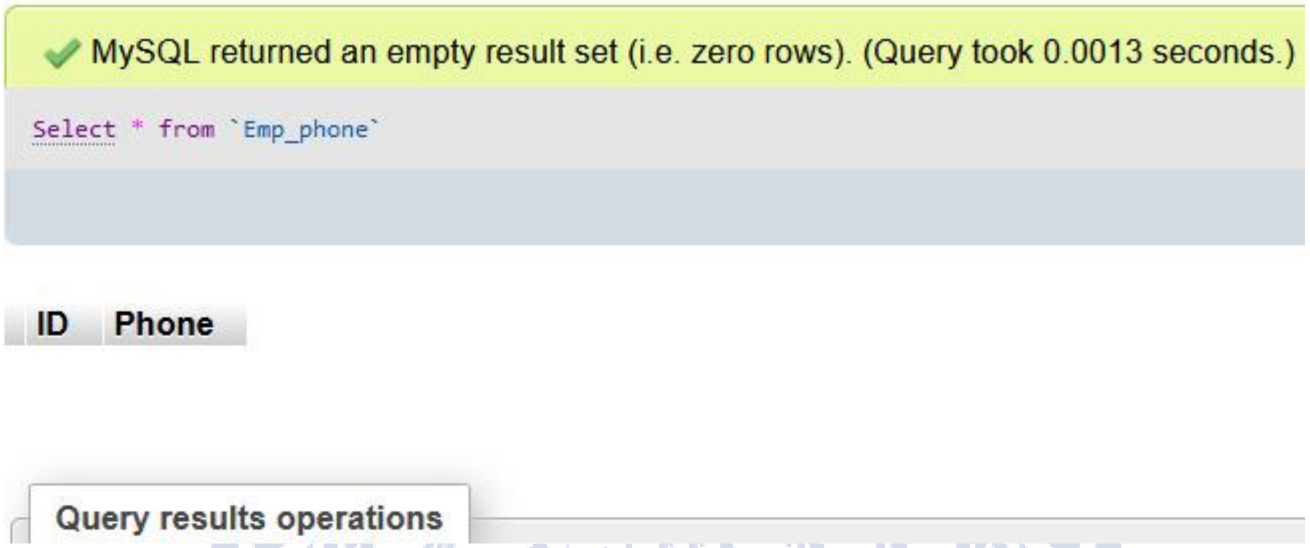
The Emp_phone entity manages contact information for employees, providing a direct link between employee IDs and phone numbers for communication purposes.

SQL Query:



```
CREATE TABLE Emp_phone (
    ID INT PRIMARY KEY,
    Phone INT
)
```

Before Insertion:

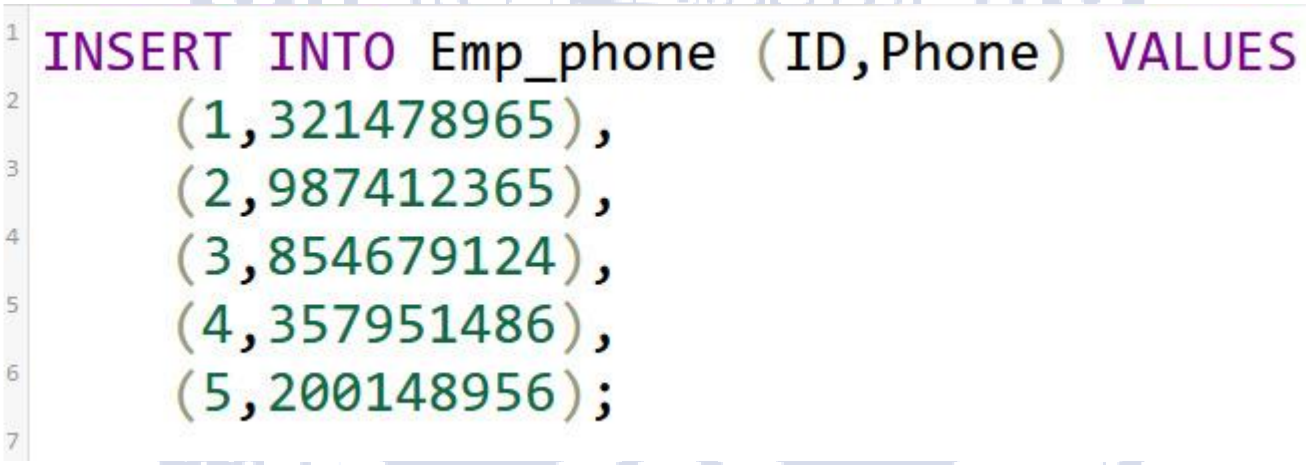


MySQL returned an empty result set (i.e. zero rows). (Query took 0.0013 seconds.)

```
Select * from `Emp_phone`
```

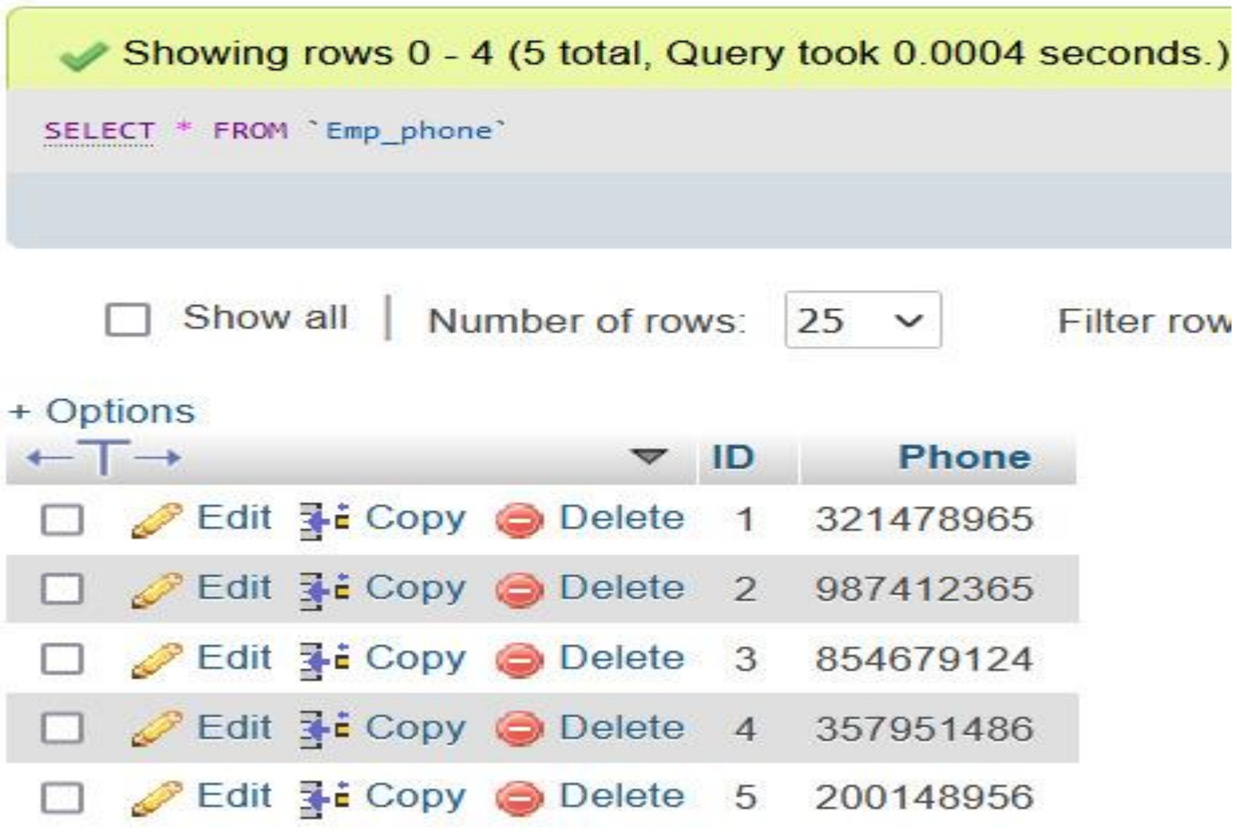
ID	Phone

Insertion:



```
INSERT INTO Emp_phone (ID,Phone) VALUES
    (1,321478965),
    (2,987412365),
    (3,854679124),
    (4,357951486),
    (5,200148956);
```

After insertion:


Showing rows 0 - 4 (5 total, Query took 0.0004 seconds.)

```
SELECT * FROM `Emp_phone`
```

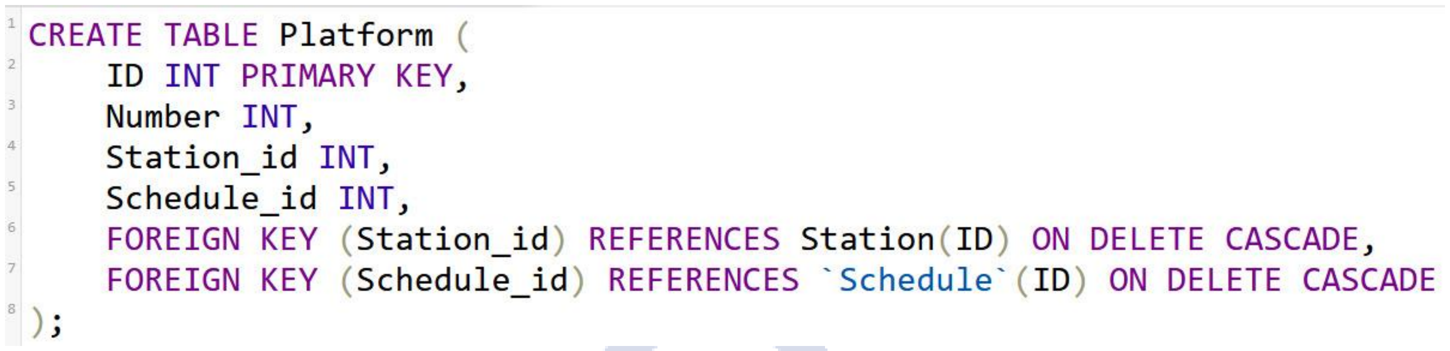
	ID	Phone
Edit Copy Delete	1	321478965
Edit Copy Delete	2	987412365
Edit Copy Delete	3	854679124
Edit Copy Delete	4	357951486
Edit Copy Delete	5	200148956

3.3. Platform

- **Attributes:**
  - ID: Unique identifier for the platform.
  - Number: Platform number within the station.
  - Station_id: Reference to the associated station.
  - Schedule_id: Reference to the associated schedule.
- **Description:**

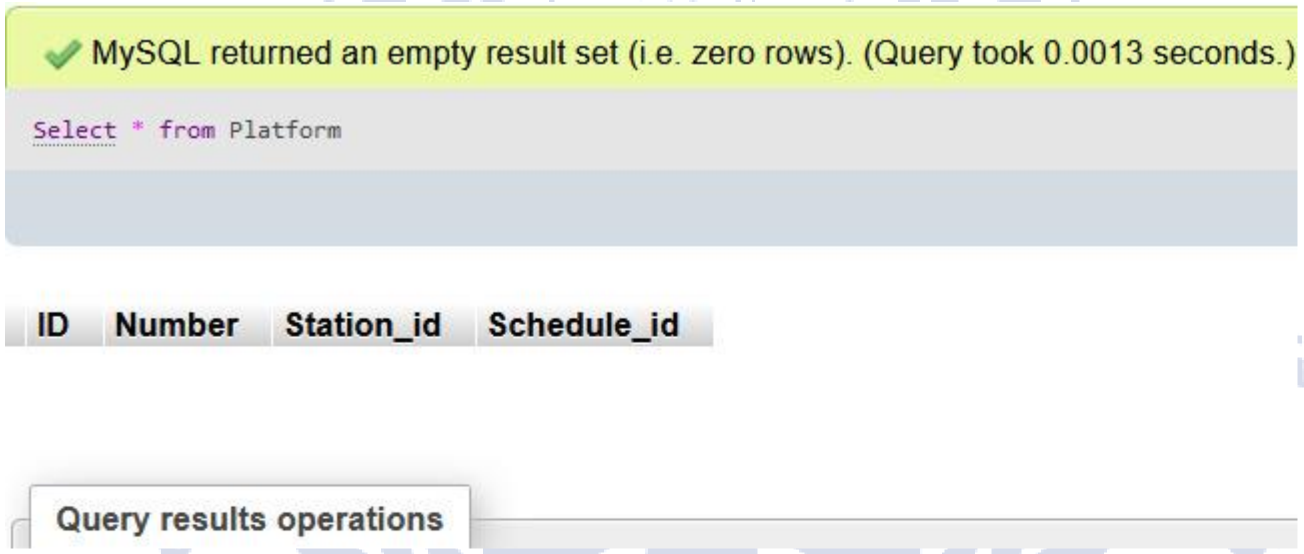
The Platform entity manages information about each platform within a station. It includes platform number, references to the associated station and schedule, facilitating seamless coordination of platform activities and schedules.

SQL Query:



```sql
CREATE TABLE Platform (
    ID INT PRIMARY KEY,
    Number INT,
    Station_id INT,
    Schedule_id INT,
    FOREIGN KEY (Station_id) REFERENCES Station(ID) ON DELETE CASCADE,
    FOREIGN KEY (Schedule_id) REFERENCES `Schedule`(ID) ON DELETE CASCADE
);
```

Before Insertion:

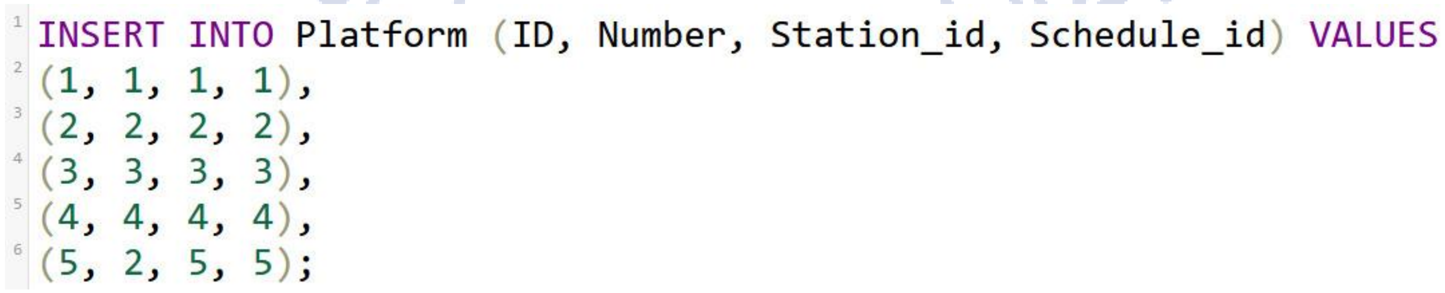


MySQL returned an empty result set (i.e. zero rows). (Query took 0.0013 seconds.)

```sql
Select * from Platform
```

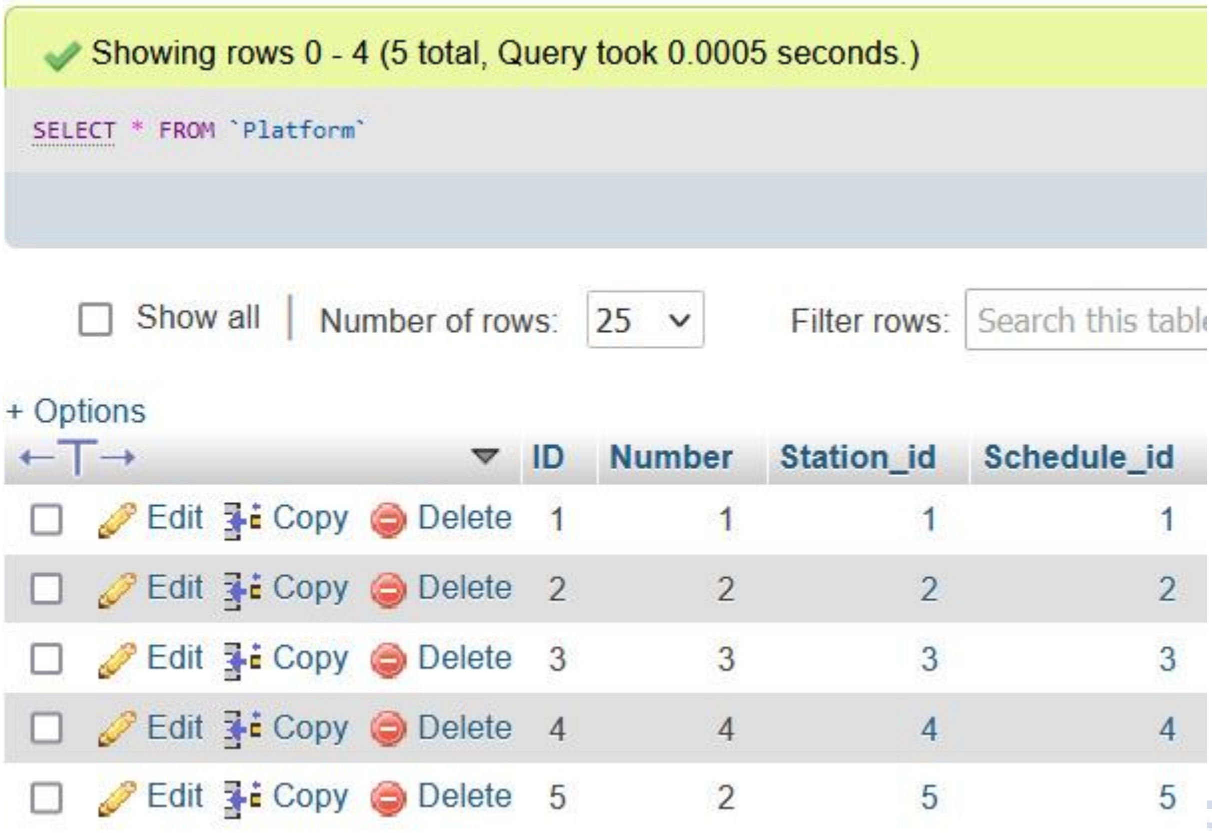
ID	Number	Station_id	Schedule_id

Insertion:



```sql
INSERT INTO Platform (ID, Number, Station_id, Schedule_id) VALUES
(1, 1, 1, 1),
(2, 2, 2, 2),
(3, 3, 3, 3),
(4, 4, 4, 4),
(5, 2, 5, 5);
```

After insertion:



Showing rows 0 - 4 (5 total, Query took 0.0005 seconds.)

```
SELECT * FROM `Platform`
```

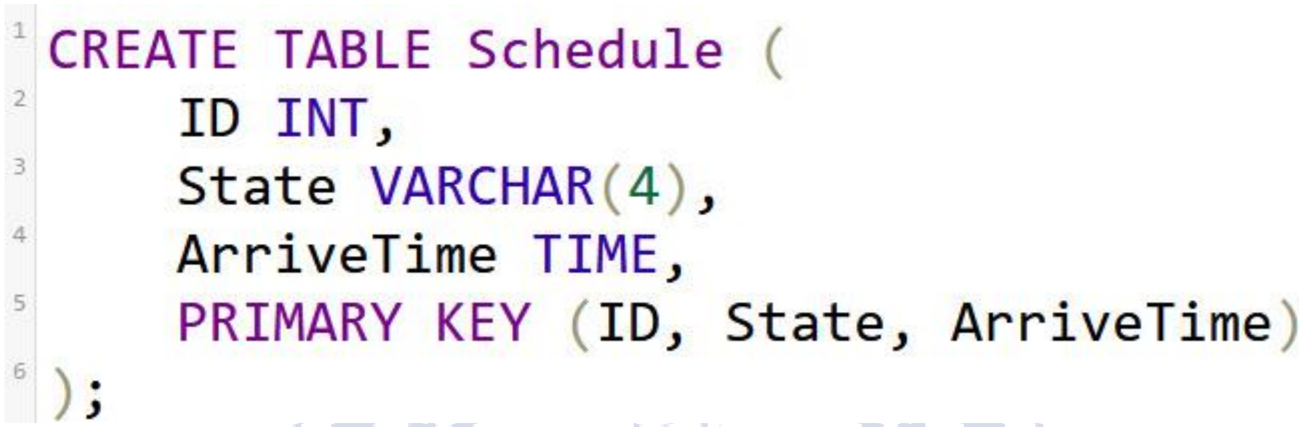
ID	Number	Station_id	Schedule_id
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	2	5	5

3.4. Schedule

- **Attributes:**
  - ID: Unique identifier for the schedule.
  - State: Current state of the schedule.
  - ArriveTime: Arrival time for the schedule.
- **Description:**

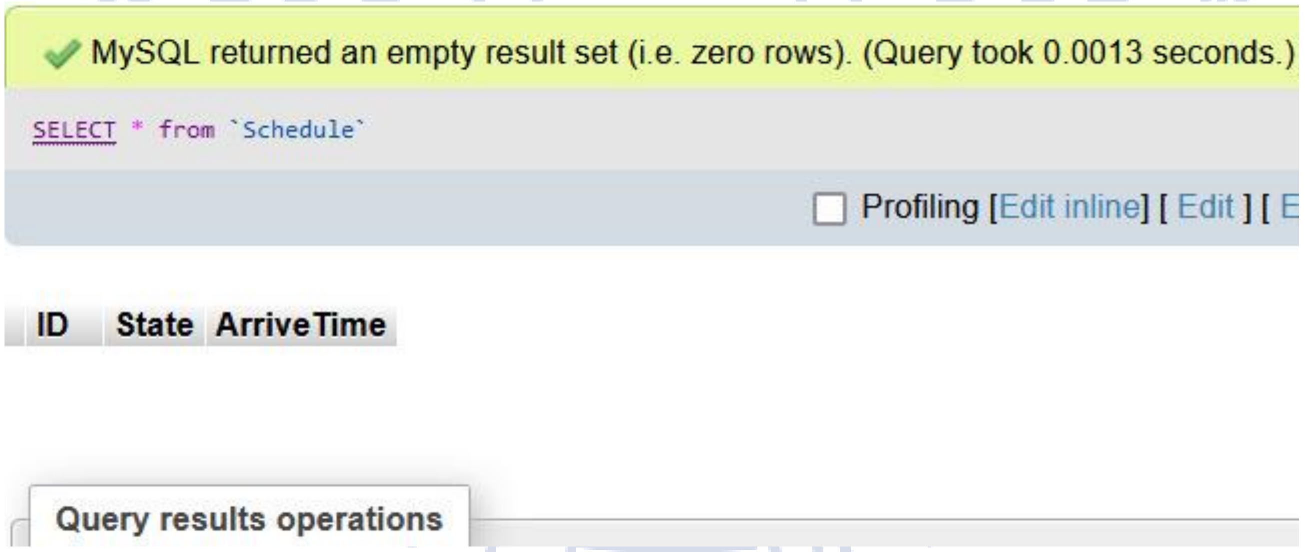
The Schedule entity captures information about train schedules, including unique identifiers, current state, and arrival times. It forms the foundation for managing train arrivals and departures.

SQL Query:



```sql
CREATE TABLE Schedule (
    ID INT,
    State VARCHAR(4),
    ArriveTime TIME,
    PRIMARY KEY (ID, State, ArriveTime)
);
```

Before Insertion:

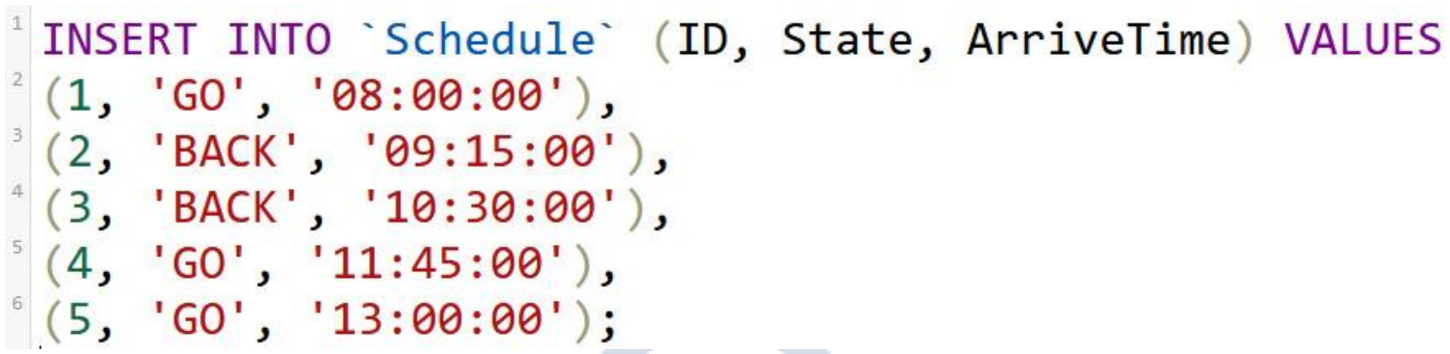


```
✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0013 seconds.)

SELECT * from `Schedule`
```

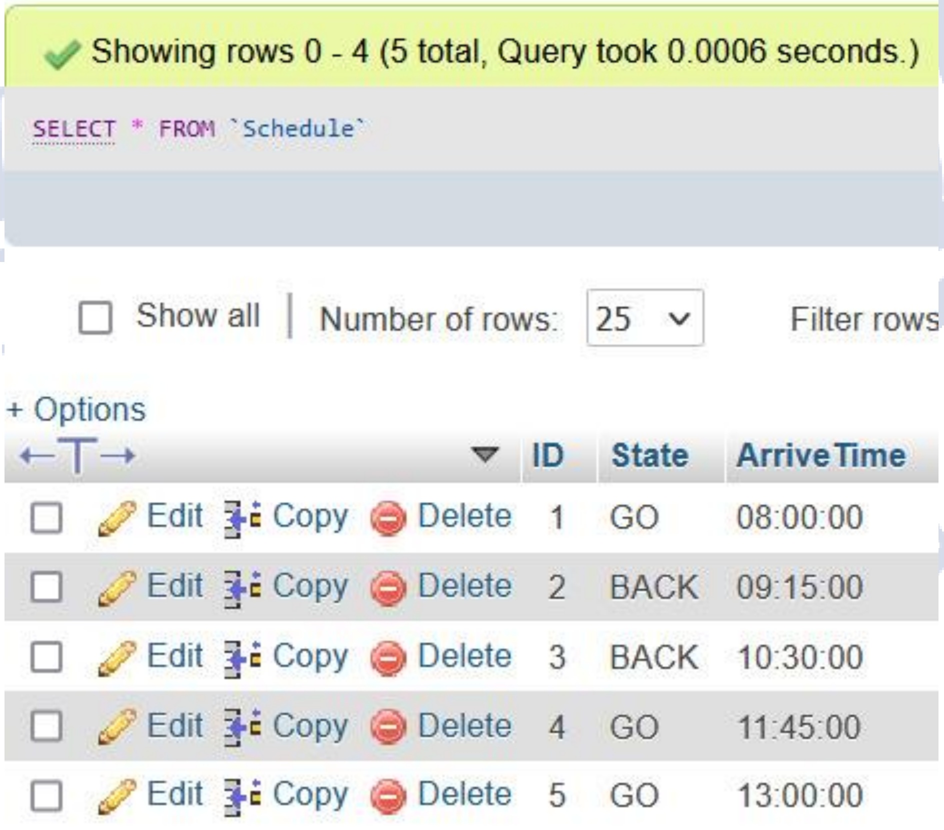
ID	State	ArriveTime

Insertion:



```sql
INSERT INTO `Schedule` (ID, State, ArriveTime) VALUES
(1, 'GO', '08:00:00'),
(2, 'BACK', '09:15:00'),
(3, 'BACK', '10:30:00'),
(4, 'GO', '11:45:00'),
(5, 'GO', '13:00:00');
```

After insertion:



Showing rows 0 - 4 (5 total, Query took 0.0006 seconds.)

```sql
SELECT * FROM `Schedule`
```

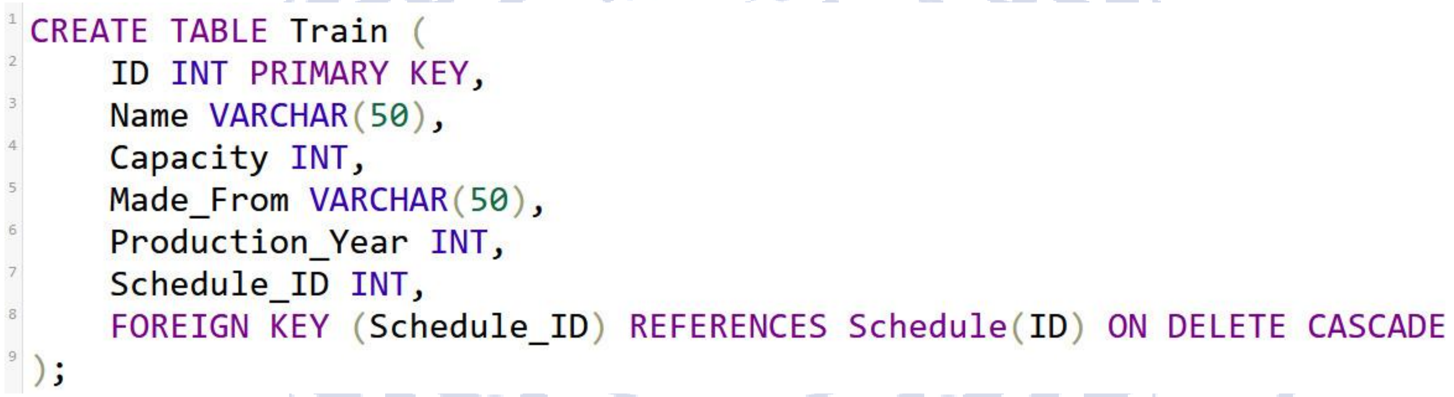
	ID	State	ArriveTime
Edit Copy Delete	1	GO	08:00:00
Edit Copy Delete	2	BACK	09:15:00
Edit Copy Delete	3	BACK	10:30:00
Edit Copy Delete	4	GO	11:45:00
Edit Copy Delete	5	GO	13:00:00

3.5. Train

- **Attributes:**
  - ID: Unique identifier for the train.
  - Name: Name of the train.
  - Capacity: Maximum capacity of the train.
  - Made_From: Material composition of the train.
  - Production_Year: Year of production of the train.
  - Schedule_ID: Reference to the associated schedule.
- **Description:**

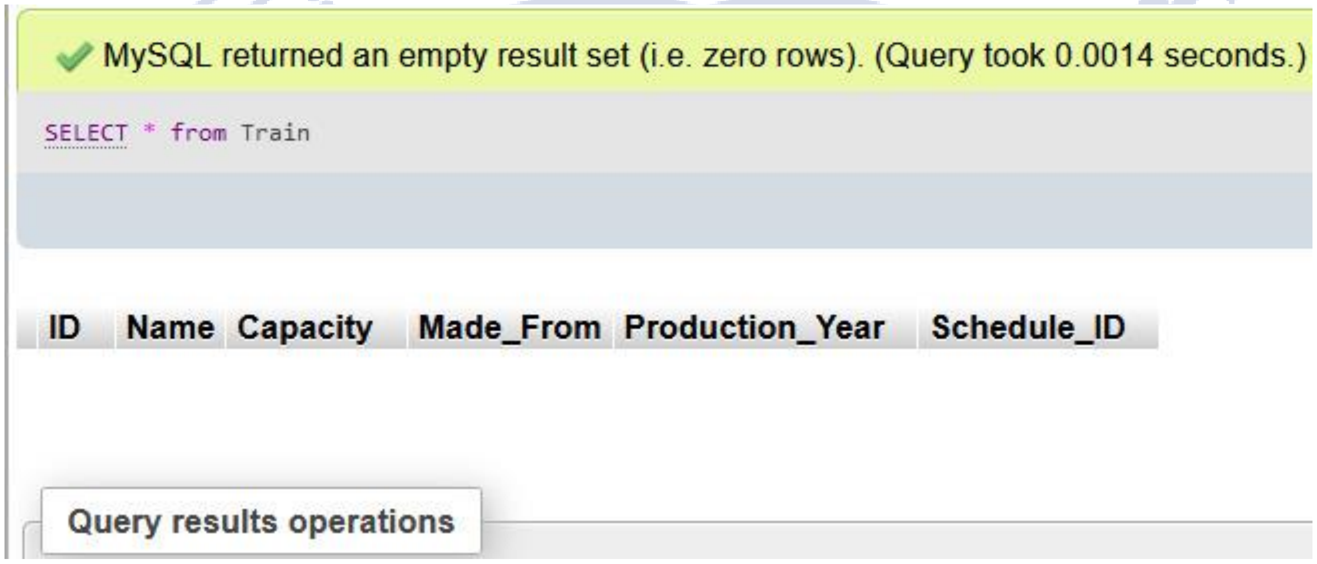
The Train entity stores comprehensive details about each train, including train name, capacity, materials used, production year, and a reference to the associated schedule. It is crucial for effective train management.

SQL Query:



```
CREATE TABLE Train (
    ID INT PRIMARY KEY,
    Name VARCHAR(50),
    Capacity INT,
    Made_From VARCHAR(50),
    Production_Year INT,
    Schedule_ID INT,
    FOREIGN KEY (Schedule_ID) REFERENCES Schedule(ID) ON DELETE CASCADE
);
```

Before Insertion:



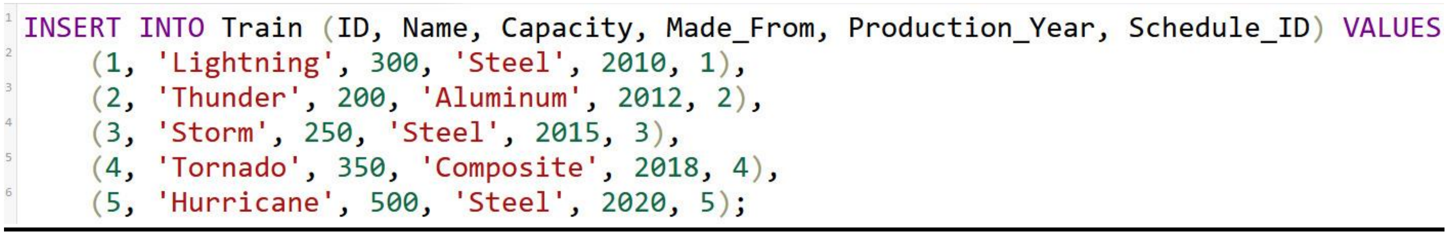
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0014 seconds.)

```
SELECT * from Train
```

ID	Name	Capacity	Made_From	Production_Year	Schedule_ID

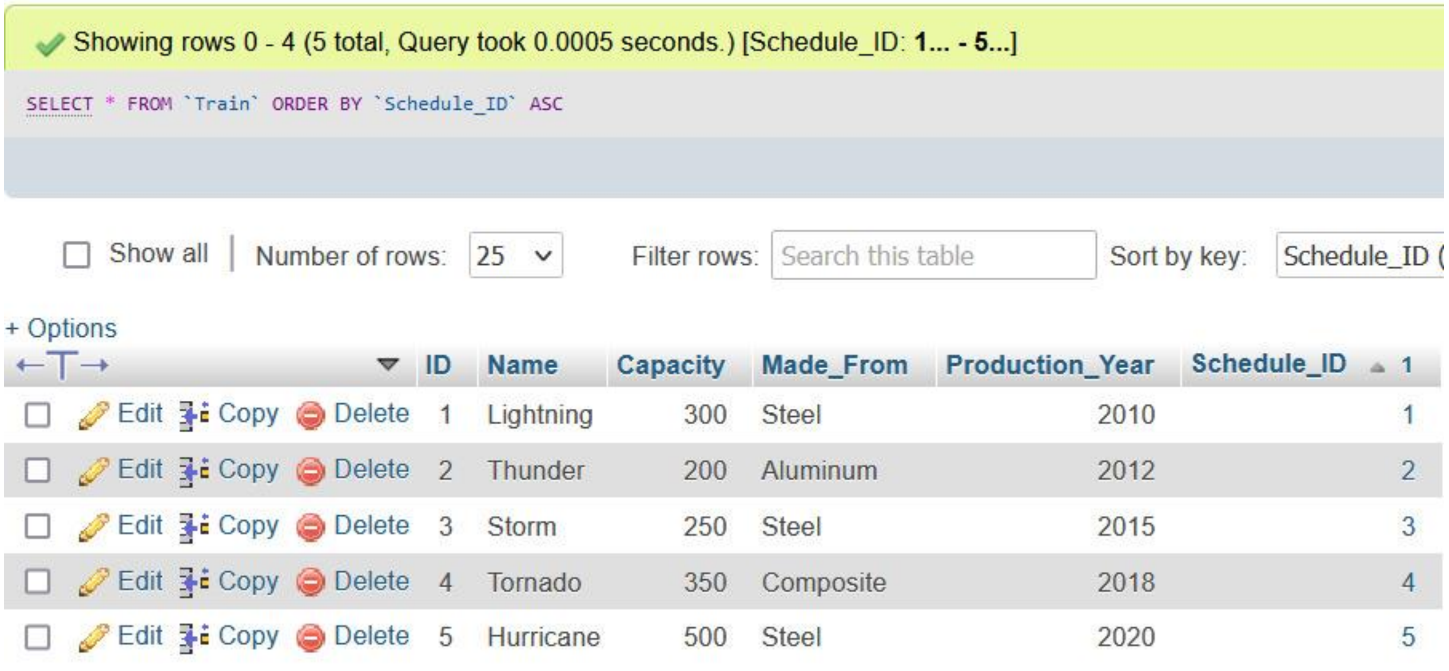
Query results operations

Insertion:



```
INSERT INTO Train (ID, Name, Capacity, Made_From, Production_Year, Schedule_ID) VALUES
    (1, 'Lightning', 300, 'Steel', 2010, 1),
    (2, 'Thunder', 200, 'Aluminum', 2012, 2),
    (3, 'Storm', 250, 'Steel', 2015, 3),
    (4, 'Tornado', 350, 'Composite', 2018, 4),
    (5, 'Hurricane', 500, 'Steel', 2020, 5);
```

After insertion:



Showing rows 0 - 4 (5 total, Query took 0.0005 seconds.) [Schedule_ID: **1... - 5...**]

```
SELECT * FROM `Train` ORDER BY `Schedule_ID` ASC
```

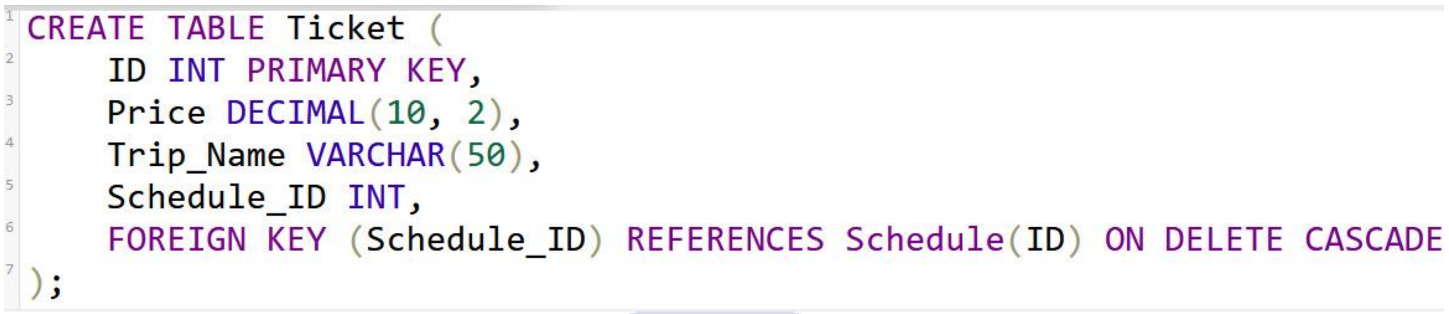
	ID	Name	Capacity	Made_From	Production_Year	Schedule_ID
Edit Copy Delete	1	Lightning	300	Steel	2010	1
Edit Copy Delete	2	Thunder	200	Aluminum	2012	2
Edit Copy Delete	3	Storm	250	Steel	2015	3
Edit Copy Delete	4	Tornado	350	Composite	2018	4
Edit Copy Delete	5	Hurricane	500	Steel	2020	5

3.6. Ticket

- **Attributes:**
  - ID: Unique identifier for the ticket.
  - Price: Price of the ticket.
  - Trip_Name: Name of the trip associated with the ticket.
  - Schedule_ID: Reference to the associated schedule.
- **Description:**

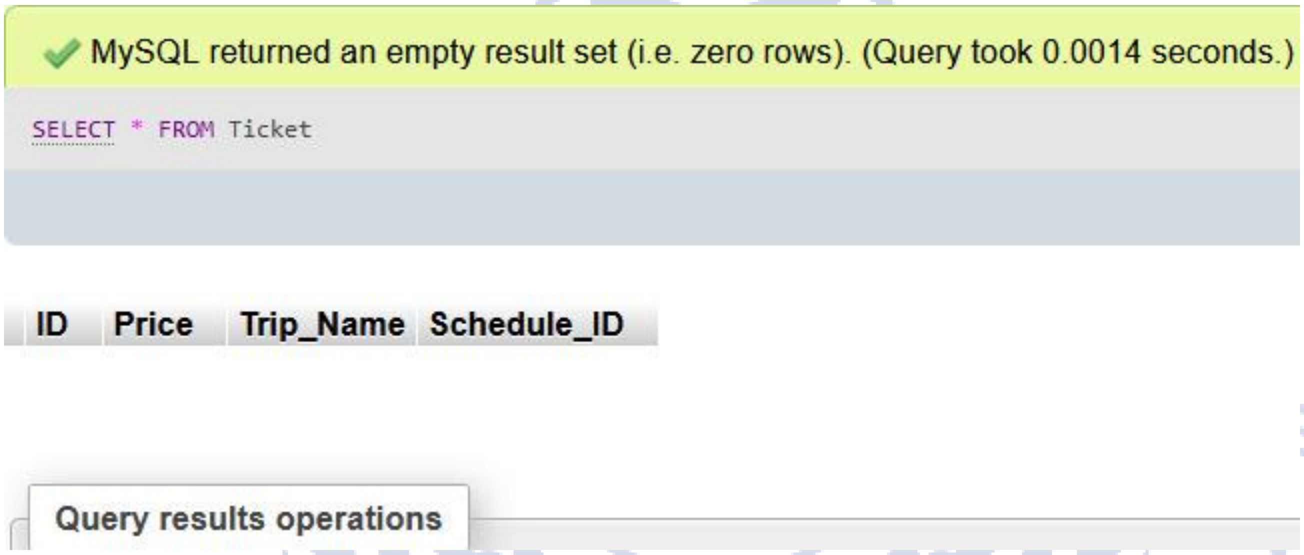
The Ticket entity manages ticket-related data, including ticket price, trip names, and references to the associated schedule. It facilitates efficient ticketing operations.

SQL Query:



```sql
CREATE TABLE Ticket (
    ID INT PRIMARY KEY,
    Price DECIMAL(10, 2),
    Trip_Name VARCHAR(50),
    Schedule_ID INT,
    FOREIGN KEY (Schedule_ID) REFERENCES Schedule(ID) ON DELETE CASCADE
);
```

Before Insertion:

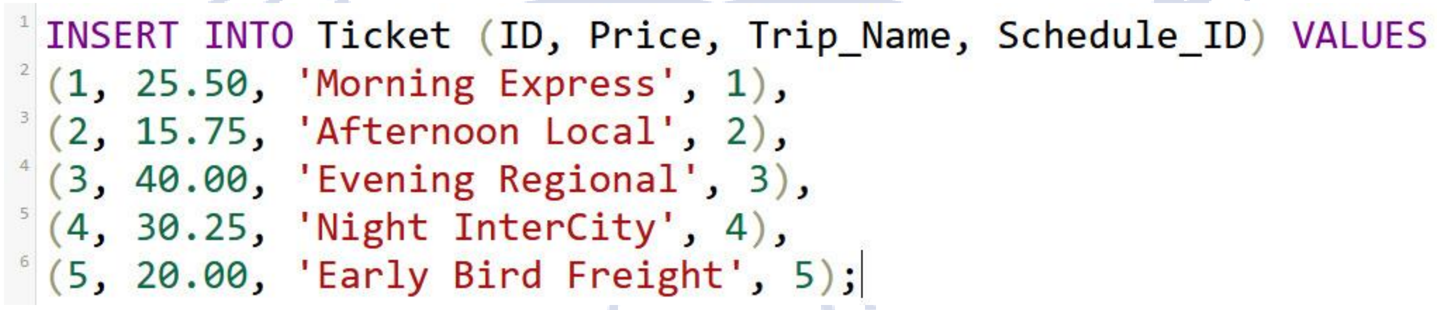


✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0014 seconds.)

```sql
SELECT * FROM Ticket
```

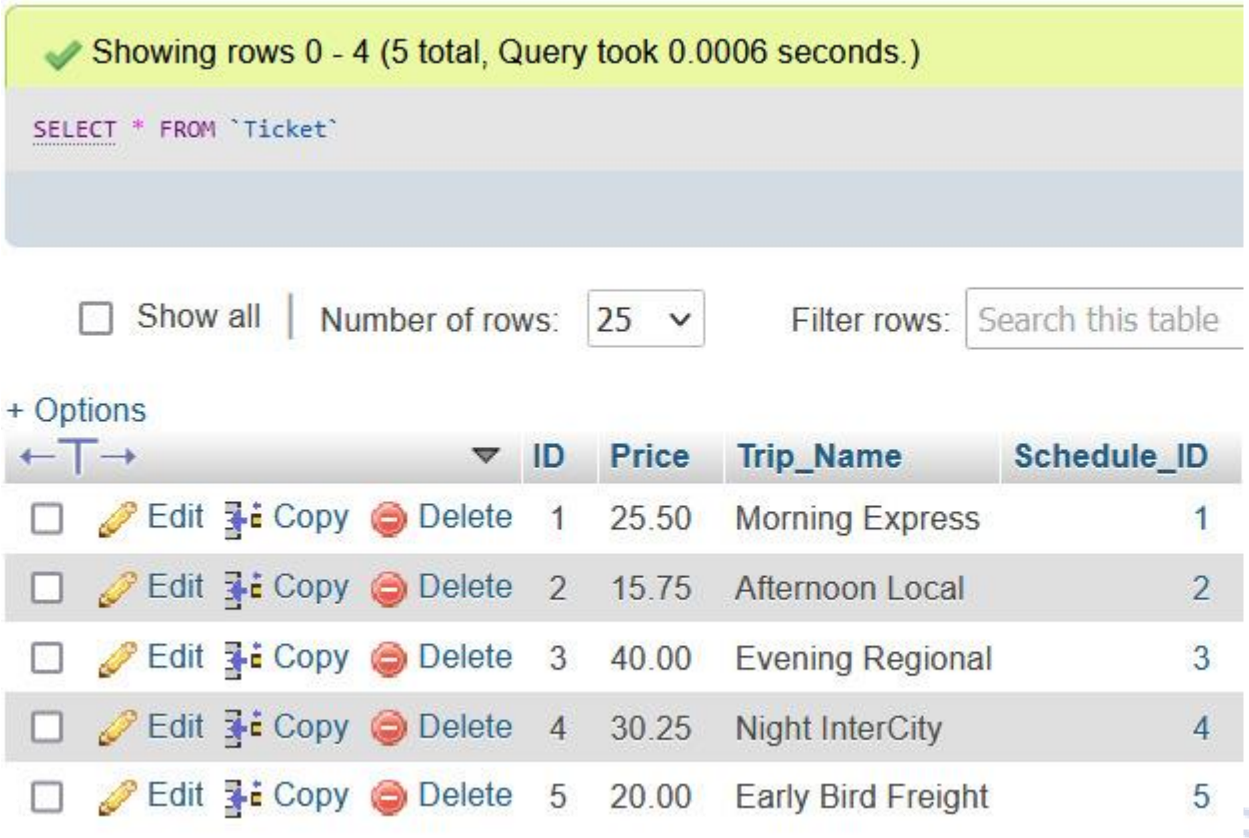
ID	Price	Trip_Name	Schedule_ID

Insertion:



```sql
INSERT INTO Ticket (ID, Price, Trip_Name, Schedule_ID) VALUES
(1, 25.50, 'Morning Express', 1),
(2, 15.75, 'Afternoon Local', 2),
(3, 40.00, 'Evening Regional', 3),
(4, 30.25, 'Night InterCity', 4),
(5, 20.00, 'Early Bird Freight', 5);
```

After insertion:


Showing rows 0 - 4 (5 total, Query took 0.0006 seconds.)

```
SELECT * FROM `Ticket`
```

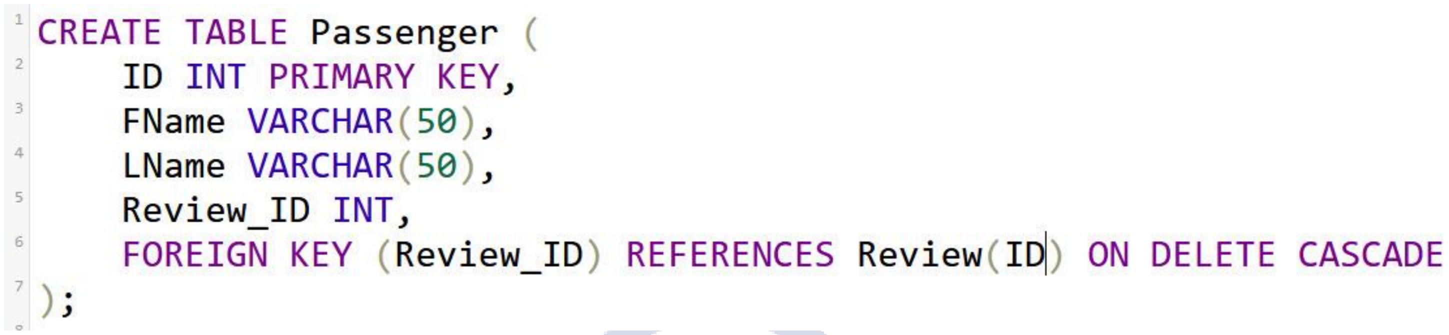
	ID	Price	Trip_Name	Schedule_ID
Edit Copy Delete	1	25.50	Morning Express	1
Edit Copy Delete	2	15.75	Afternoon Local	2
Edit Copy Delete	3	40.00	Evening Regional	3
Edit Copy Delete	4	30.25	Night InterCity	4
Edit Copy Delete	5	20.00	Early Bird Freight	5

3.7. Passenger:

- **Attributes:**
  - ID: Unique identifier for the passenger.
  - FName: First name of the passenger.
  - LName: Last name of the passenger.
  - Review_ID: Reference to the associated review.
- **Description:**

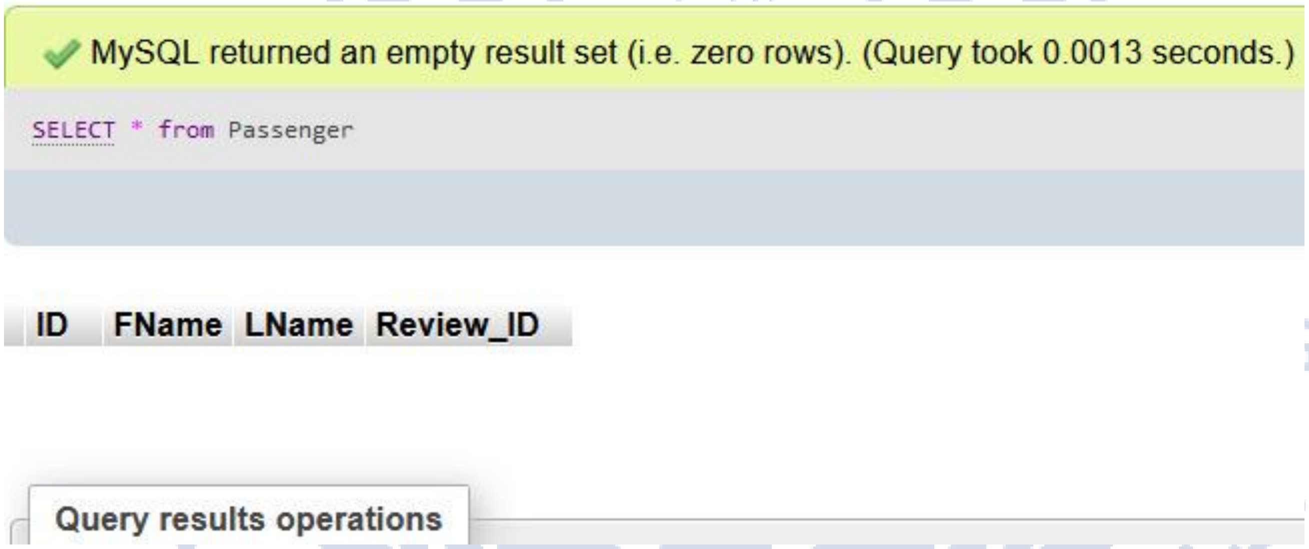
The Passenger entity stores details about each passenger, including names and a reference to the associated review. This allows for personalized passenger experiences and feedback tracking.

SQL Query:



```sql
CREATE TABLE Passenger (
    ID INT PRIMARY KEY,
    FName VARCHAR(50),
    LName VARCHAR(50),
    Review_ID INT,
    FOREIGN KEY (Review_ID) REFERENCES Review(ID) ON DELETE CASCADE
);
```

Before Insertion:

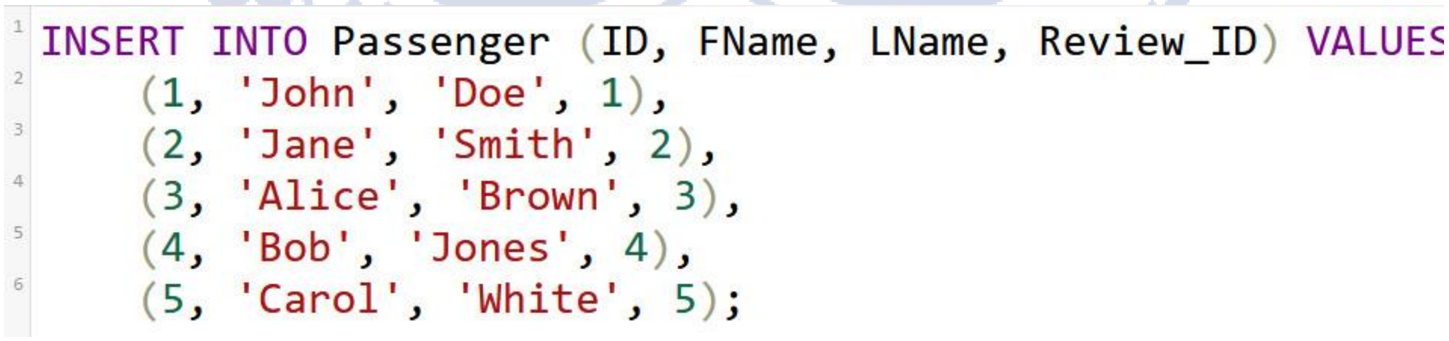


MySQL returned an empty result set (i.e. zero rows). (Query took 0.0013 seconds.)

```sql
SELECT * from Passenger
```

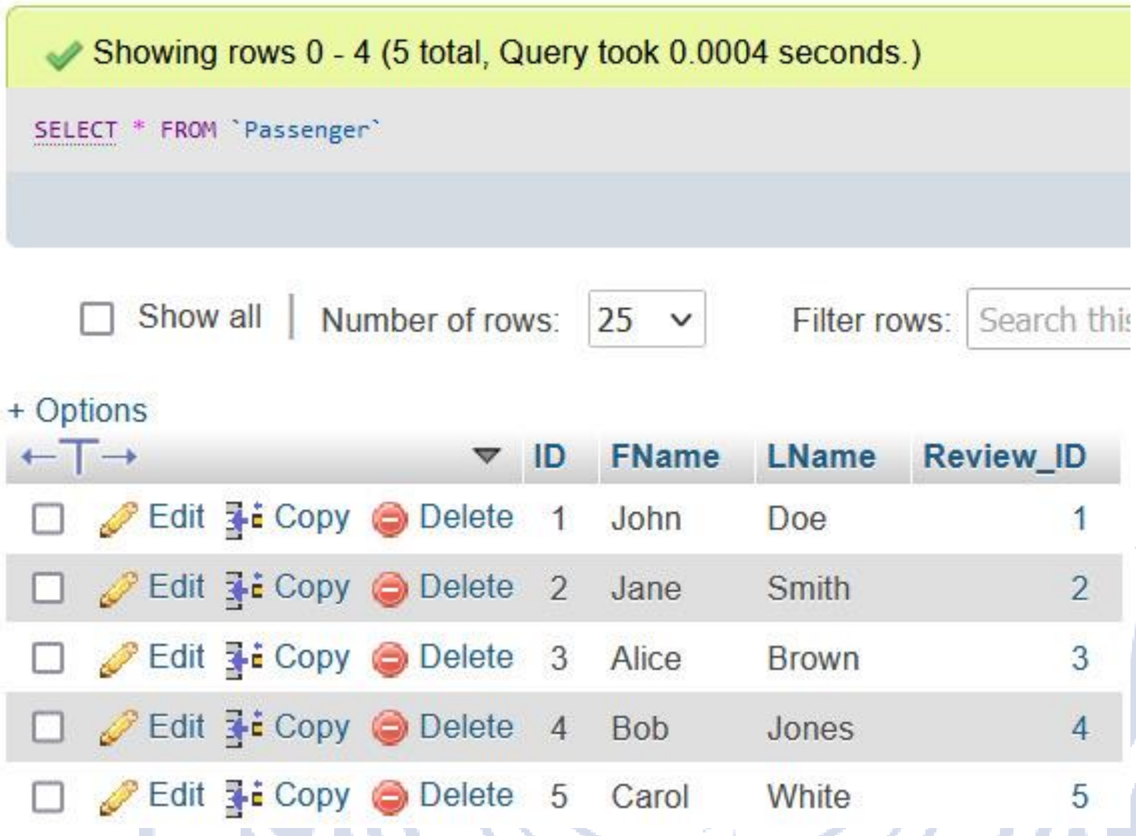
ID	FName	LName	Review_ID

Insertion:



```sql
INSERT INTO Passenger (ID, FName, LName, Review_ID) VALUES
    (1, 'John', 'Doe', 1),
    (2, 'Jane', 'Smith', 2),
    (3, 'Alice', 'Brown', 3),
    (4, 'Bob', 'Jones', 4),
    (5, 'Carol', 'White', 5);
```

After insertion:



Showing rows 0 - 4 (5 total, Query took 0.0004 seconds.)

```
SELECT * FROM `Passenger`
```

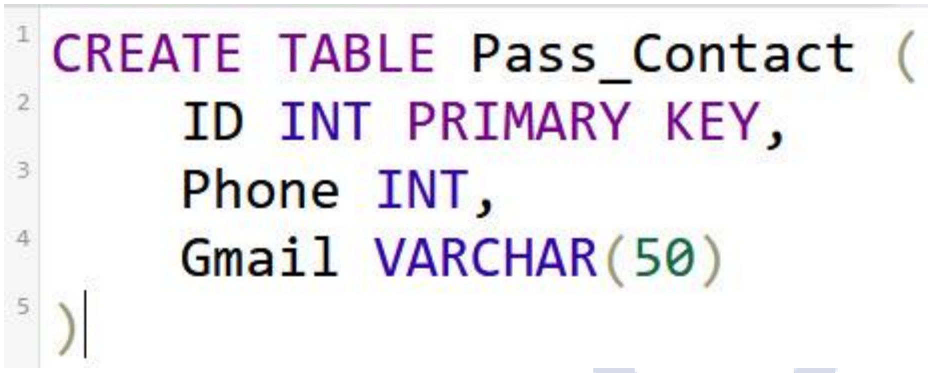
ID	FName	LName	Review_ID
1	John	Doe	1
2	Jane	Smith	2
3	Alice	Brown	3
4	Bob	Jones	4
5	Carol	White	5

& Pass_Contact (multi value in passenger)

- **Attributes:**
  - ID: Unique identifier for passenger contact information.
  - Phone: Phone number of the passenger.
  - Gmail: Gmail address of the passenger.
- **Description:**

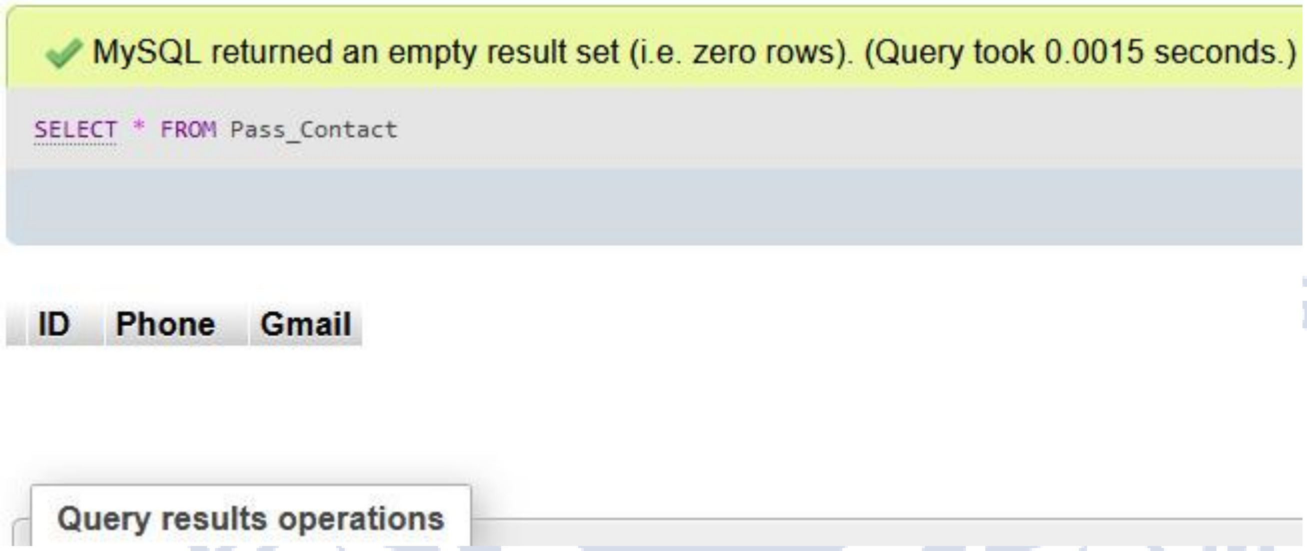
The Pass_Contact entity manages contact details for passengers, including phone numbers and Gmail addresses, enhancing communication and service personalization.

SQL Query:



```
CREATE TABLE Pass_Contact (
    ID INT PRIMARY KEY,
    Phone INT,
    Gmail VARCHAR(50)
)
```

Before Insertion:

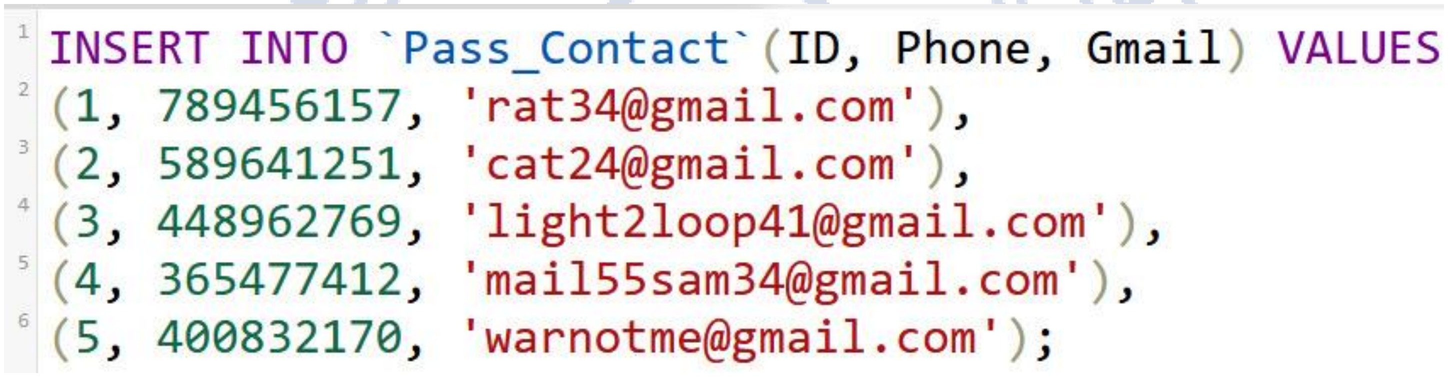


```
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0015 seconds.)

SELECT * FROM Pass_Contact
```

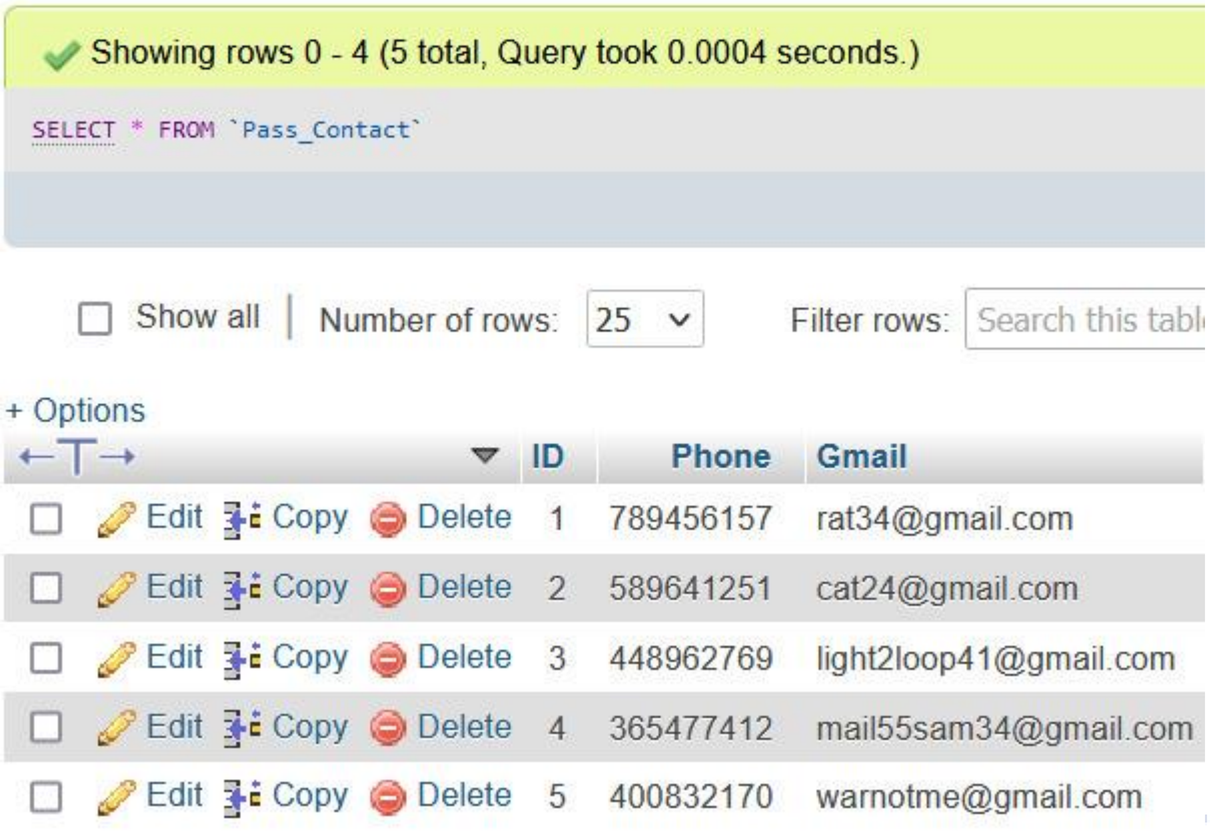
ID	Phone	Gmail

Insertion:



```
INSERT INTO `Pass_Contact`(ID, Phone, Gmail) VALUES
(1, 789456157, 'rat34@gmail.com'),
(2, 589641251, 'cat24@gmail.com'),
(3, 448962769, 'light2loop41@gmail.com'),
(4, 365477412, 'mail55sam34@gmail.com'),
(5, 400832170, 'warnotme@gmail.com');
```

After insertion:


Showing rows 0 - 4 (5 total, Query took 0.0004 seconds.)

```
SELECT * FROM `Pass_Contact`
```

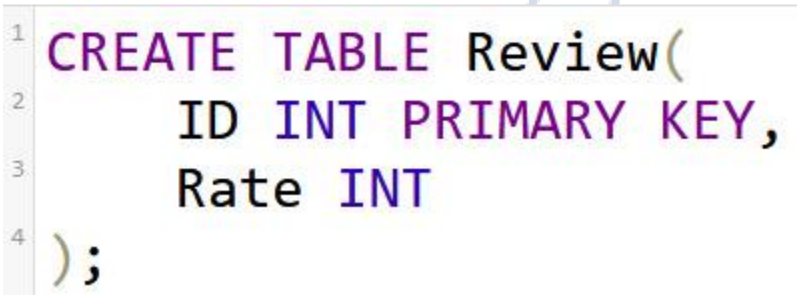
ID	Phone	Gmail
1	789456157	rat34@gmail.com
2	589641251	cat24@gmail.com
3	448962769	light2loop41@gmail.com
4	365477412	mail55sam34@gmail.com
5	400832170	warnotme@gmail.com

3.8. Review

- **Attributes:**
    - ID: Unique identifier for the review.
    - Rate: Rating given in the review.
- **Description:**

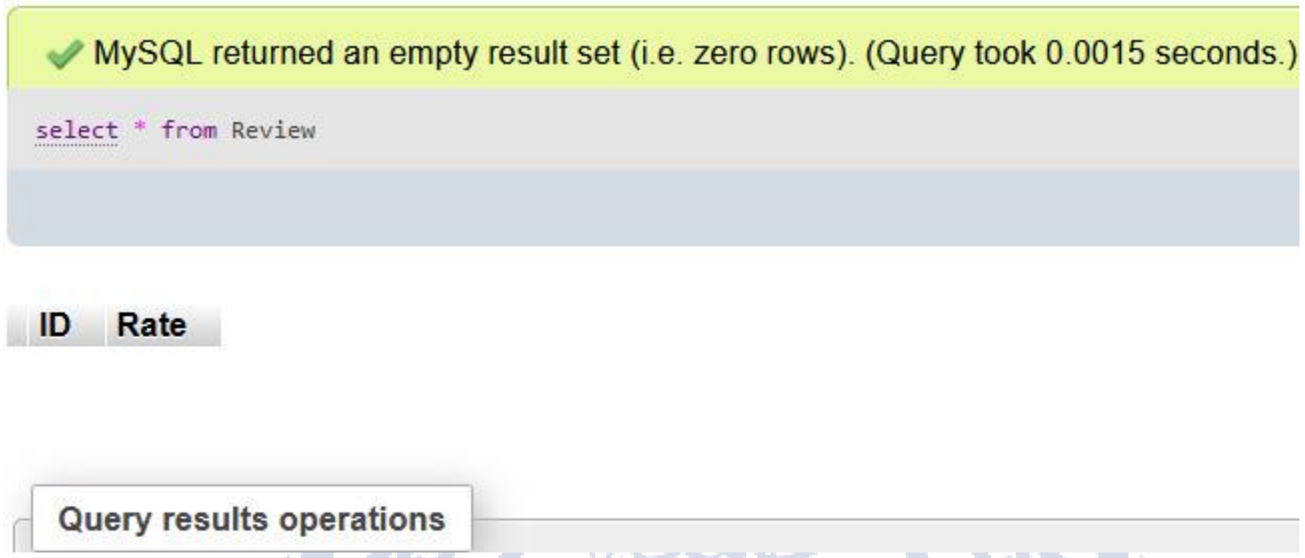
The Review entity manages passenger reviews, capturing review IDs and ratings. This facilitates continuous improvement in service quality based on passenger feedback.

SQL Query:



```
CREATE TABLE Review(
    ID INT PRIMARY KEY,
    Rate INT
);
```

Before Insertion:

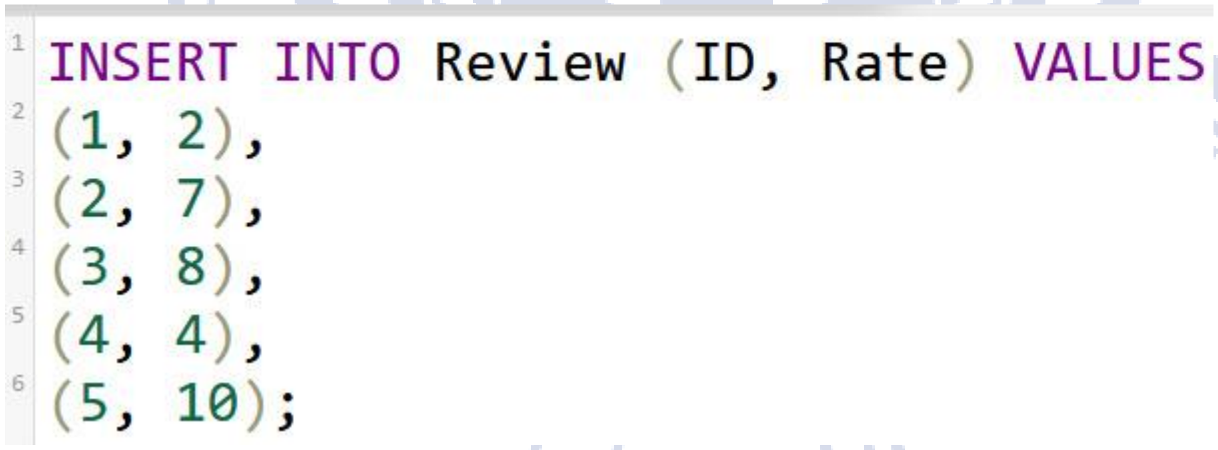


MySQL returned an empty result set (i.e. zero rows). (Query took 0.0015 seconds.)

```
select * from Review
```

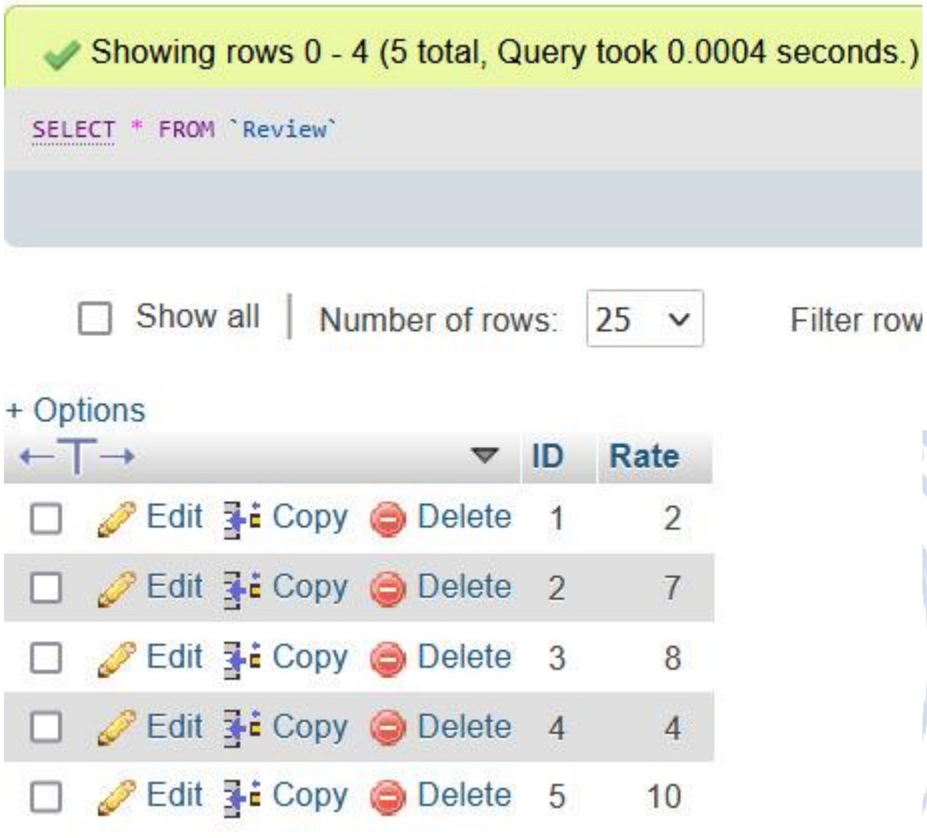
ID	Rate

Insertion:



```
INSERT INTO Review (ID, Rate) VALUES
(1, 2),
(2, 7),
(3, 8),
(4, 4),
(5, 10);
```

After insertion:


Showing rows 0 - 4 (5 total, Query took 0.0004 seconds.)

```
SELECT * FROM `Review`
```

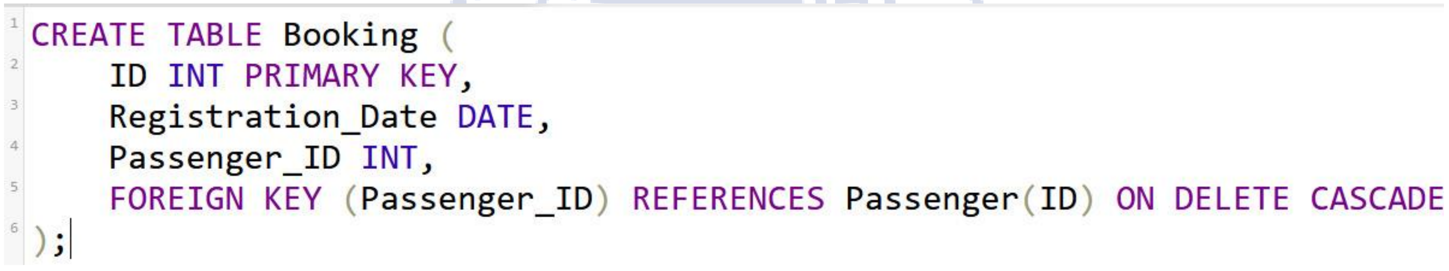
ID	Rate
1	2
2	7
3	8
4	4
5	10

3.9. Booking

- **Attributes:**
    - ID: Unique identifier for the booking.
    - Registration_Date: Date of registration for the booking.
    - Passenger_ID: Reference to the associated passenger.
- **Description:**

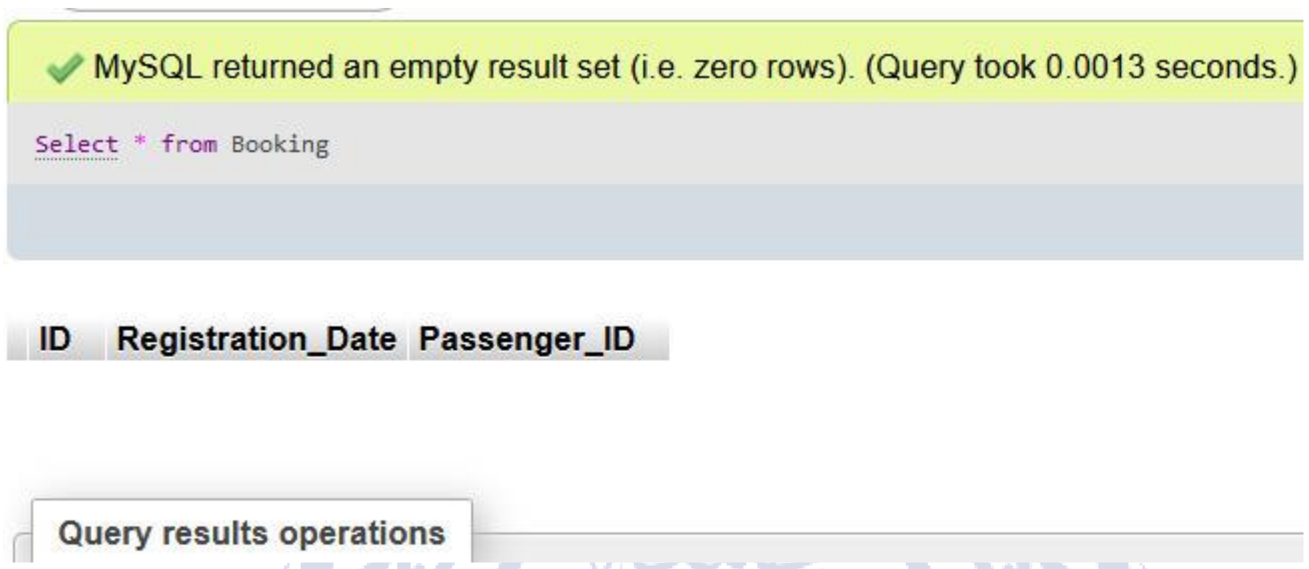
The Booking entity manages booking information, including booking IDs, registration dates, and references to the associated passenger. It is essential for tracking and coordinating passenger travel plans.

SQL Query:



```
CREATE TABLE Booking (
    ID INT PRIMARY KEY,
    Registration_Date DATE,
    Passenger_ID INT,
    FOREIGN KEY (Passenger_ID) REFERENCES Passenger(ID) ON DELETE CASCADE
);
```

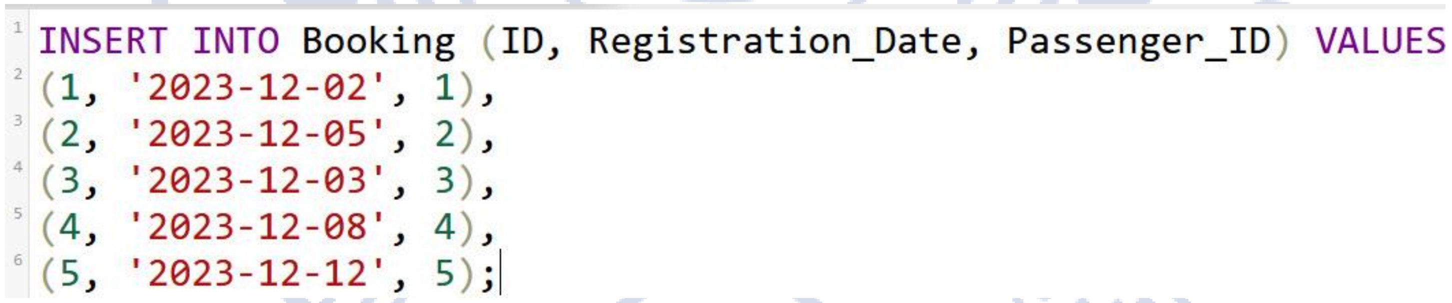
Before Insertion:


MySQL returned an empty result set (i.e. zero rows). (Query took 0.0013 seconds.)

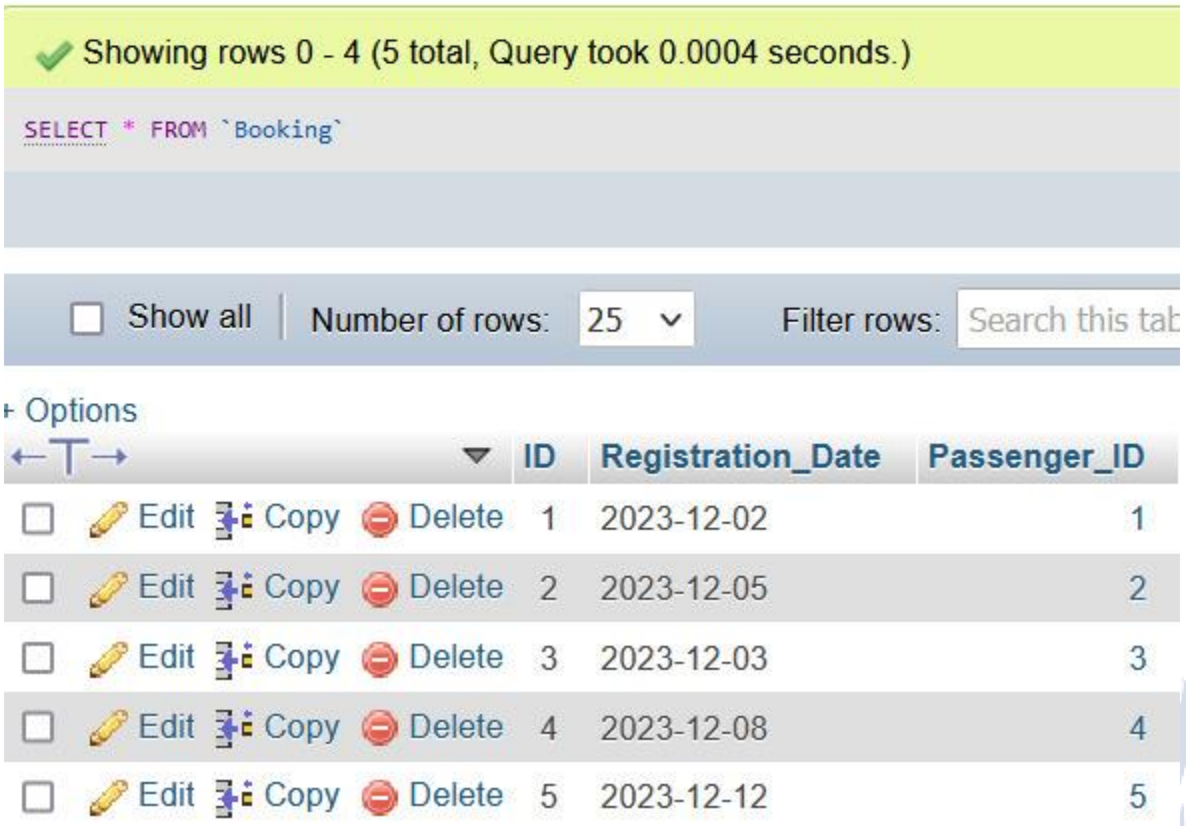
```
Select * from Booking
```

ID	Registration_Date	Passenger_ID

Insertion:


```
INSERT INTO Booking (ID, Registration_Date, Passenger_ID) VALUES
(1, '2023-12-02', 1),
(2, '2023-12-05', 2),
(3, '2023-12-03', 3),
(4, '2023-12-08', 4),
(5, '2023-12-12', 5);
```

After insertion:


✓ Showing rows 0 - 4 (5 total, Query took 0.0004 seconds.)

```
SELECT * FROM `Booking`
```

Show all | Number of rows: 25 | Filter rows: Search this tab

+ Options

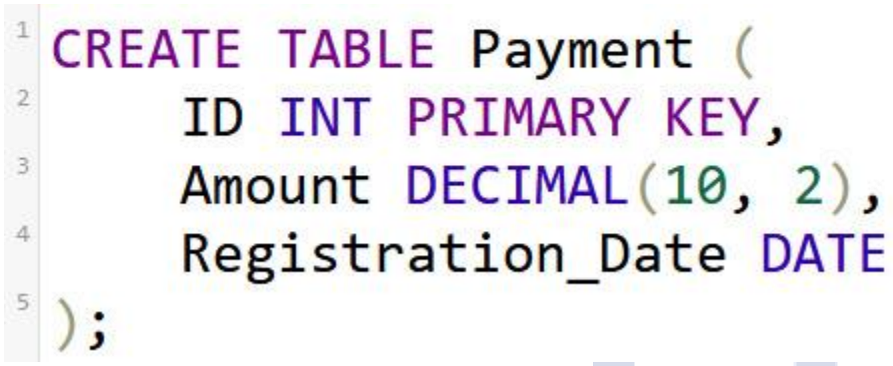
	ID	Registration_Date	Passenger_ID
Edit Copy Delete	1	2023-12-02	1
Edit Copy Delete	2	2023-12-05	2
Edit Copy Delete	3	2023-12-03	3
Edit Copy Delete	4	2023-12-08	4
Edit Copy Delete	5	2023-12-12	5

3.10. Payment

- **Attributes:**
  - ID: Unique identifier for the payment.
  - Amount: Amount of the payment.
  - Registration_Date: Date of registration for the payment.
- **Description:**

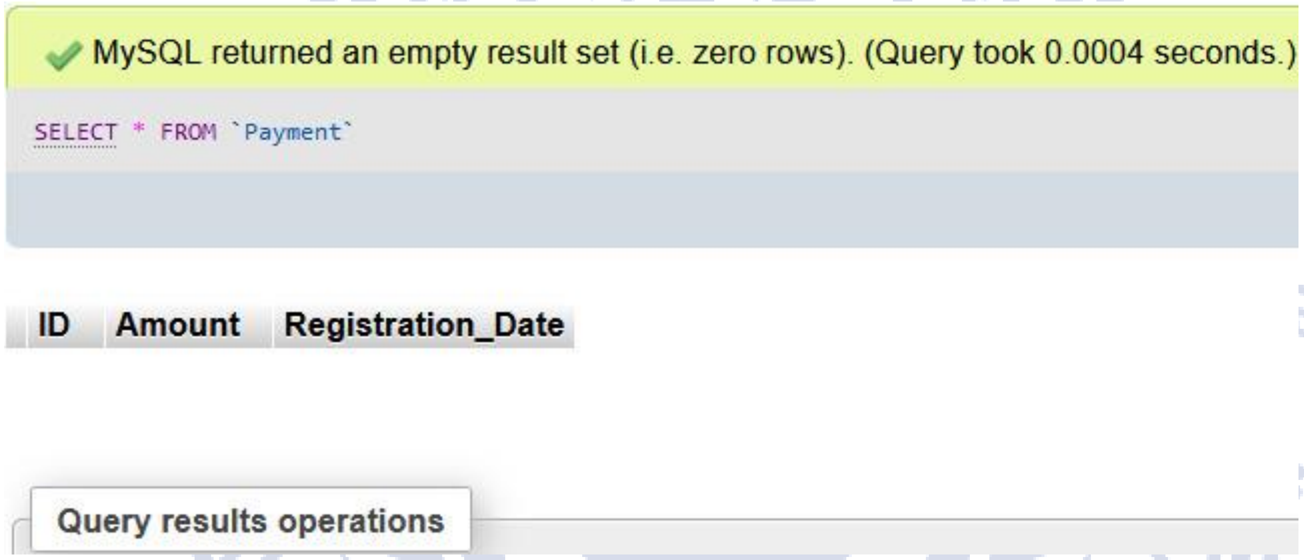
The Payment entity tracks financial transactions, capturing payment IDs, amounts, and registration dates. This ensures transparency and accountability in financial operations.

SQL Query:



```sql
CREATE TABLE Payment (
    ID INT PRIMARY KEY,
    Amount DECIMAL(10, 2),
    Registration_Date DATE
);
```

Before Insertion:

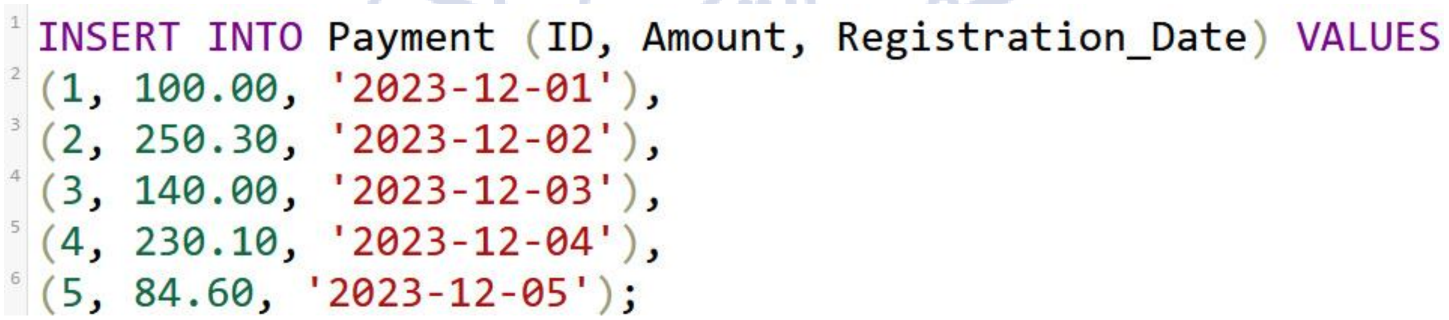


```
✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0004 seconds.)

SELECT * FROM `Payment`
```

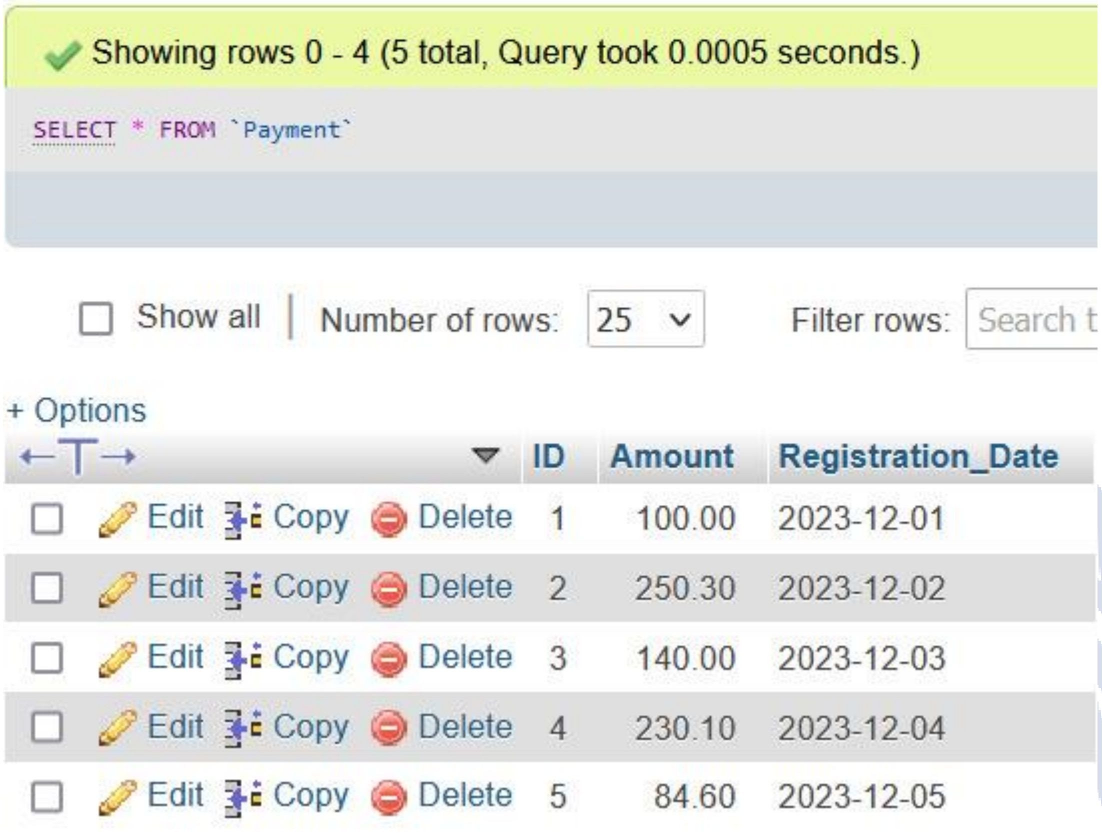
ID	Amount	Registration_Date

Insertion:



```sql
INSERT INTO Payment (ID, Amount, Registration_Date) VALUES
(1, 100.00, '2023-12-01'),
(2, 250.30, '2023-12-02'),
(3, 140.00, '2023-12-03'),
(4, 230.10, '2023-12-04'),
(5, 84.60, '2023-12-05');
```

After insertion:



Showing rows 0 - 4 (5 total, Query took 0.0005 seconds.)

```
SELECT * FROM `Payment`
```

Show all | Number of rows: 25 Filter rows: Search t

+ Options

	ID	Amount	Registration_Date
Edit Copy Delete	1	100.00	2023-12-01
Edit Copy Delete	2	250.30	2023-12-02
Edit Copy Delete	3	140.00	2023-12-03
Edit Copy Delete	4	230.10	2023-12-04
Edit Copy Delete	5	84.60	2023-12-05

4. Relationships Between Entities

Each entity is interconnected through well-defined relationships, ensuring the system's relational integrity.

In a relational database, the relationships between tables are typically handled through the use of foreign keys. The SQL statements for creating the tables provided in your initial input already include the foreign keys necessary to represent the relationships shown in the ERD.

Station - Platform Relationship:

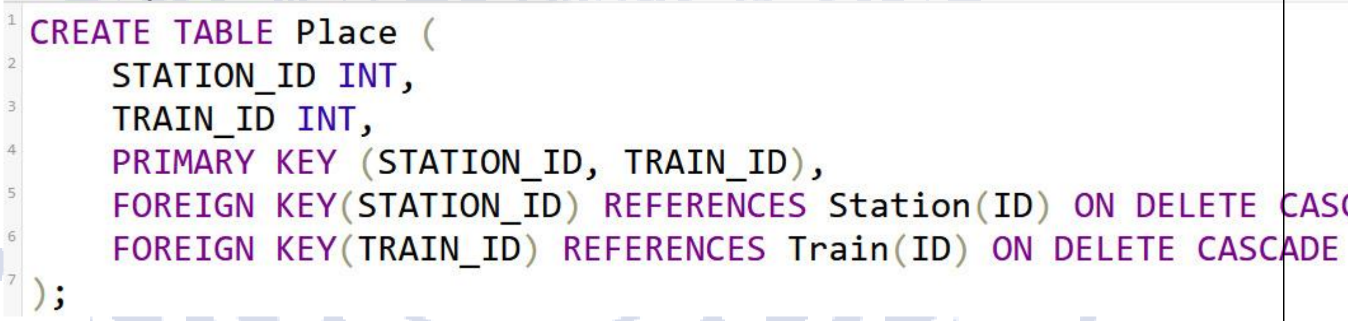
- **Relationship Type:** One-to-Many
- **Description:** Each train station (Station entity) can have multiple platforms (Platform entity). The foreign key constraint in the Platform entity references the Station entity, establishing a one-to-many relationship. This connection allows for efficient platform management within each station.

Station - Employee Relationship:

- **Relationship Type:** One-to-Many
- **Description:** Each train station (Station entity) can have multiple employees (Employee entity). The foreign key constraint in the Employee entity references the Station entity, establishing a one-to-many relationship. This connection facilitates transparent management of the station workforce.

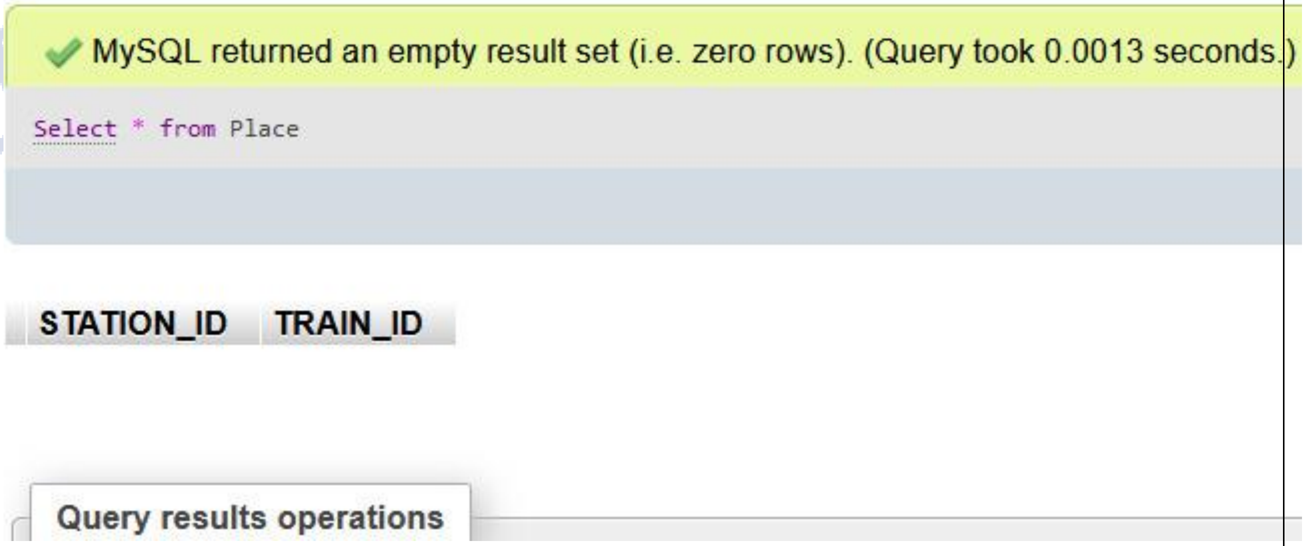
Station - Train Relationship:

- **Relationship Type:** Many-to-Many → **(new taple)**
- **Description:** Train stations (Station entity) and trains (Train entity) are connected through a many-to-many relationship. This relationship is established by an associative table that holds foreign key references to both the Station and Train entities. In this configuration, a station can be associated with multiple trains, and a train can be linked to multiple stations. This flexible relationship accommodates the diverse scenarios within the train station system, allowing for effective organization and connectivity between stations and trains.
- **SQL Query:**



```
CREATE TABLE Place (
    STATION_ID INT,
    TRAIN_ID INT,
    PRIMARY KEY (STATION_ID, TRAIN_ID),
    FOREIGN KEY(STATION_ID) REFERENCES Station(ID) ON DELETE CASC
    FOREIGN KEY(TRAIN_ID) REFERENCES Train(ID) ON DELETE CASCADE
);
```

- **Before Insertion:**

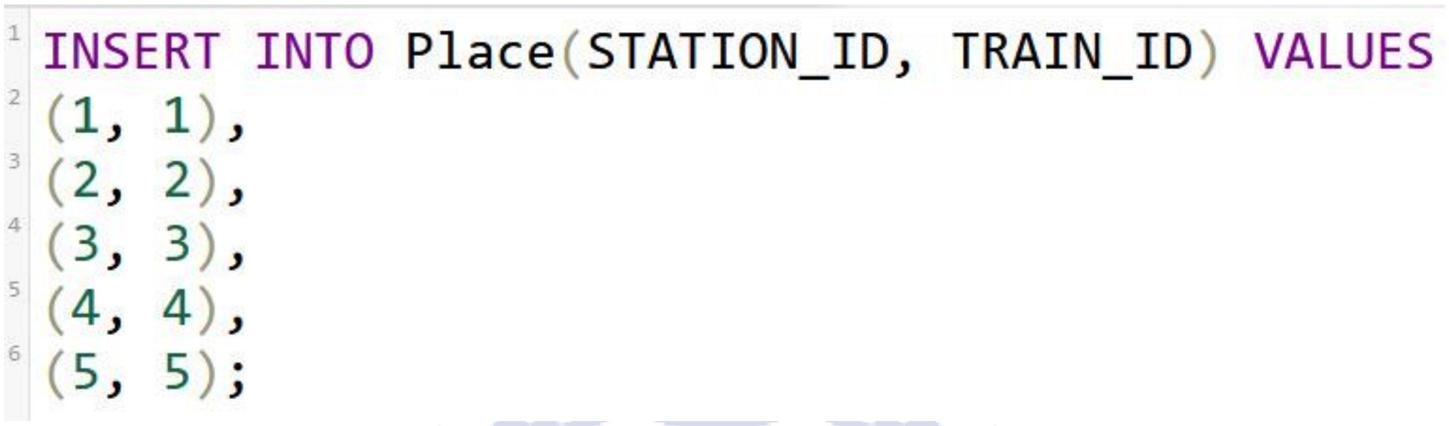


```
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0013 seconds.)

Select * from Place
```

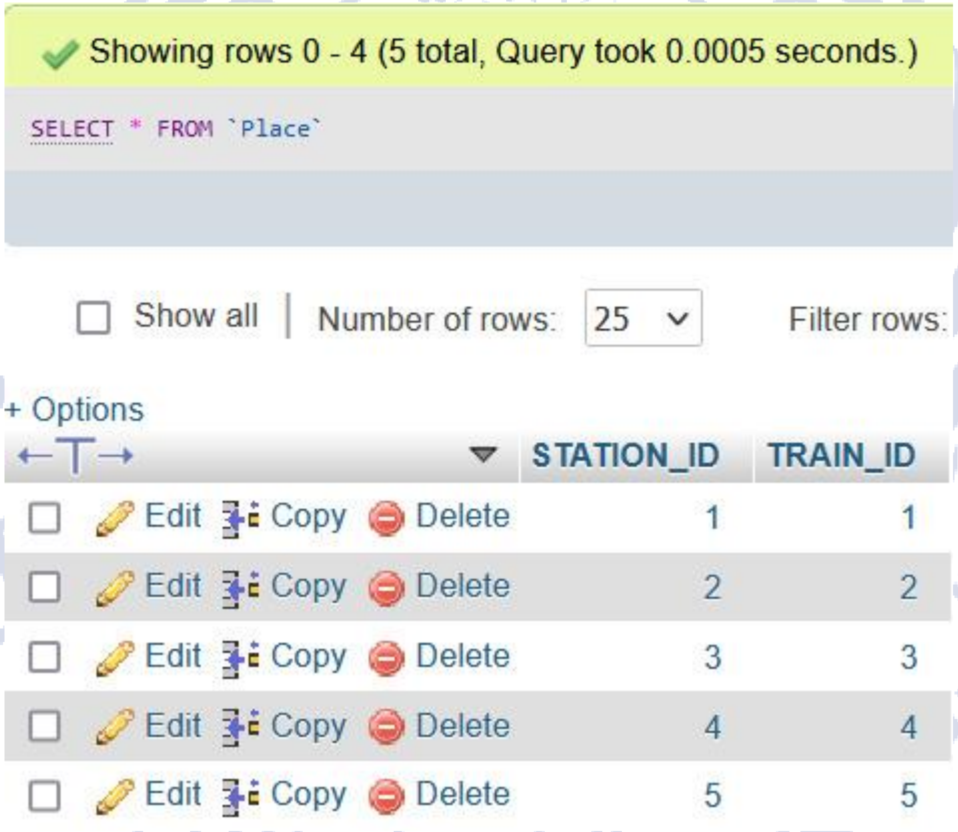
STATION_ID	TRAIN_ID

- **Insertion:**



```
INSERT INTO Place(STATION_ID, TRAIN_ID) VALUES
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(5, 5);
```

- **After insertion:**



Showing rows 0 - 4 (5 total, Query took 0.0005 seconds.)

```
SELECT * FROM `Place`
```

	STATION_ID	TRAIN_ID
Edit Copy Delete	1	1
Edit Copy Delete	2	2
Edit Copy Delete	3	3
Edit Copy Delete	4	4
Edit Copy Delete	5	5

Platform - Schedule Relationship:

- **Relationship Type:** Many-to-One
- **Description:** Multiple platforms (Platform entity) can be associated with a single schedule (Schedule entity). The foreign key constraint in the Platform entity references the Schedule entity, establishing a many-to-one relationship. This connection allows for coordinated scheduling across multiple platforms within a station.

Train - Schedule Relationship:

- **Relationship Type:** Many-to-One
- **Description:** Multiple trains (Train entity) can be associated with a single schedule (Schedule entity). The foreign key constraint in the Train entity references the Schedule entity, establishing a many-to-one relationship. This connection enables effective management of train schedules.

Review - Passenger Relationship:

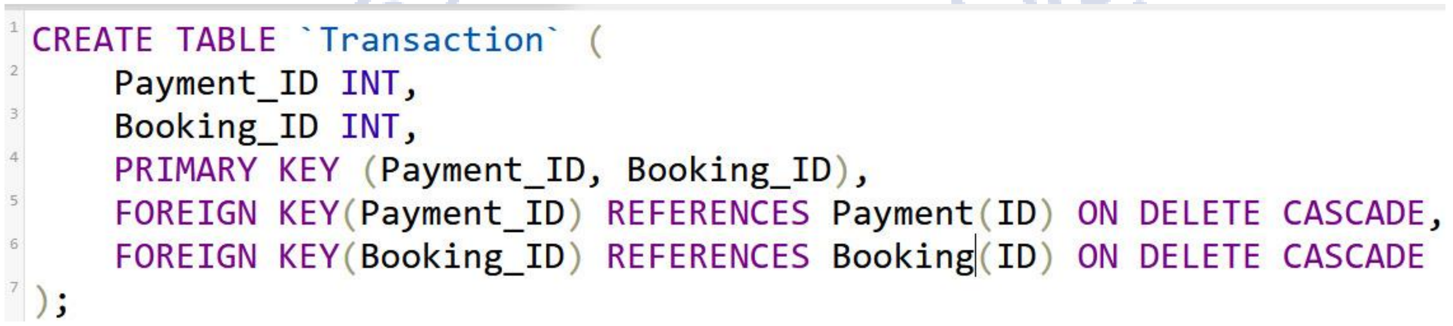
- **Relationship Type:** One-to-Many
- **Description:** Each review (Review entity) can be associated with multiple passengers (Passenger entity). The foreign key constraint in the Passenger entity references the Review entity, establishing a one-to-many relationship. This connection allows for multiple passengers to provide feedback through reviews

Booking - Passenger Relationship:

- **Relationship Type:** Many-to-One
- **Description:** Multiple bookings (Booking entity) can be associated with a single passenger (Passenger entity). The foreign key constraint in the Booking entity references the Passenger entity, establishing a many-to-one relationship. This connection allows for efficient tracking and coordination of passenger travel plans

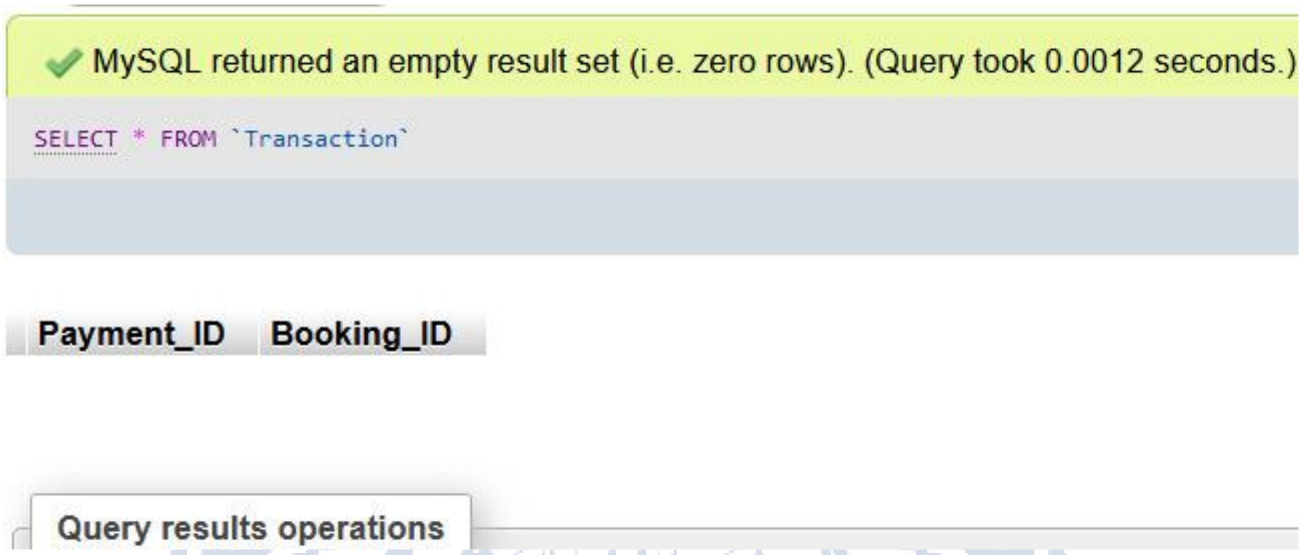
Payment - Booking Relationship:

- **Relationship Type:** Many-to-Many → **(new taple)**
- **Description:** Payments (Payment entity) and bookings (Booking entity) engage in a many-to-many relationship. This relationship is facilitated by an associative table, which holds foreign key references to both the Payment and Booking entities. In a many-to-many scenario, multiple payments can be associated with multiple bookings, and vice versa. This dynamic connection allows for a comprehensive and flexible link between payments and bookings within the train station system, accommodating various transactional scenarios.
- **SQL Query:**



```
CREATE TABLE `Transaction` (
    Payment_ID INT,
    Booking_ID INT,
    PRIMARY KEY (Payment_ID, Booking_ID),
    FOREIGN KEY(Payment_ID) REFERENCES Payment(ID) ON DELETE CASCADE,
    FOREIGN KEY(Booking_ID) REFERENCES Booking(ID) ON DELETE CASCADE
);
```

- **Before Insertion:**



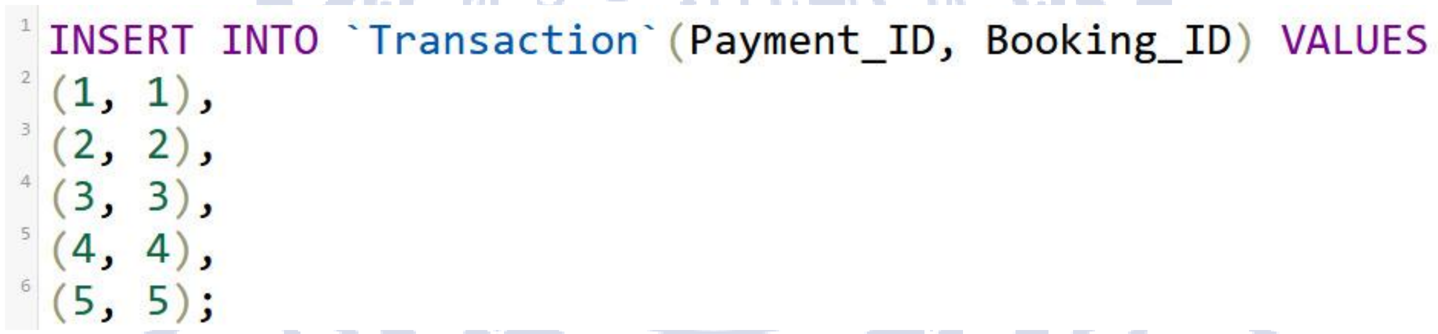
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0012 seconds.)

```
SELECT * FROM `Transaction`
```

Payment_ID	Booking_ID

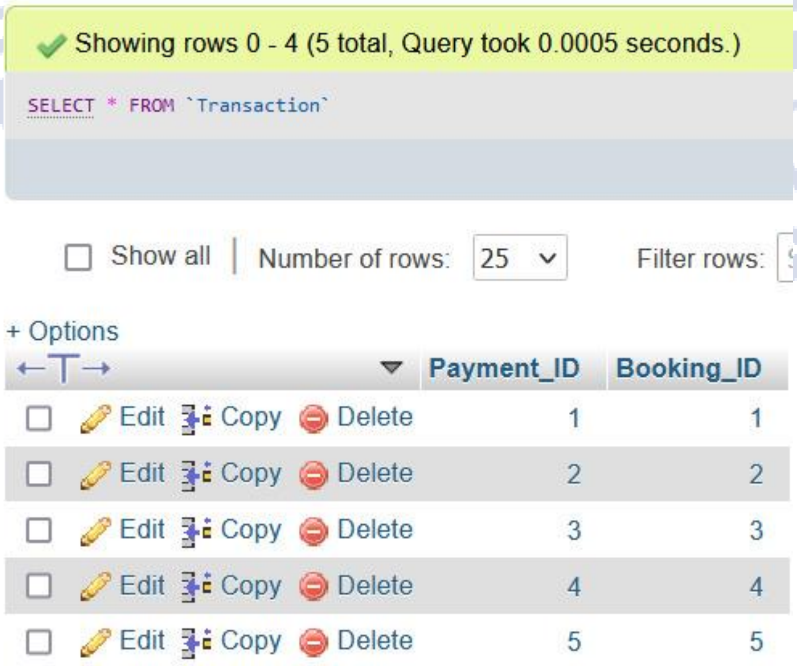
Query results operations

- **Insertion:**



```
INSERT INTO `Transaction`(Payment_ID, Booking_ID) VALUES
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(5, 5);
```

- **After insertion:**



Showing rows 0 - 4 (5 total, Query took 0.0005 seconds.)

```
SELECT * FROM `Transaction`
```

Show all | Number of rows: 25 Filter rows:

+ Options

	Payment_ID	Booking_ID
Edit Copy Delete	1	1
Edit Copy Delete	2	2
Edit Copy Delete	3	3
Edit Copy Delete	4	4
Edit Copy Delete	5	5

5. Data Integrity and Constraints

Data integrity and constraints are fundamental pillars of a robust database schema, forming the cornerstone for dependable, precise, and consistent data management within the Train Station System. Through the meticulous implementation of well-defined rules and limitations, the database guarantees that the information it stores is reliable, valid, and serves as the authoritative source for the seamless functioning of the train station operations.

Primary Keys:

In the Train Station System database, each entity is equipped with a primary key—a unique identifier that ensures the distinctiveness of each record. These primary keys thwart the occurrence of duplicate entries and establish a definitive reference for inter-table relationships. For instance, a StationID functions as the unique identifier for each train station, while a ScheduleID serves as a distinctive identifier for each schedule in the system.

Foreign Keys:

Foreign keys play a pivotal role in maintaining referential integrity between tables, linking records across different entities and preserving the interconnected relationships. In the context of the Train Station System, the Booking table might employ a PassengerID as a foreign key to reference the corresponding Passenger entity, ensuring that relationships between entities, such as passengers and bookings, are upheld and coherent.

6. Conclusion

In conclusion, the Train Station System database stands as a meticulously designed and interconnected framework, adeptly addressing the complex dynamics of a multifaceted train station ecosystem. The implementation of robust data integrity measures, characterized by well-defined primary and foreign keys, establishes a foundation of reliability, accuracy, and consistency within the system.

Through the unique identification provided by primary keys, each entity within the database is distinctly identified, mitigating the risk of duplicate entries and facilitating precise relationships between tables. The Train Station System's reliance on primary keys, such as StationID and ScheduleID, exemplifies a commitment to data accuracy and a clear reference for effective data management.

Furthermore, foreign keys play a crucial role in maintaining referential integrity, ensuring that relationships between different entities, such as passengers and bookings, are not only preserved but also coherent. The integration of foreign keys, like PassengerID linking to the Passenger entity, exemplifies a seamless approach to managing interconnected data, reflecting the dynamic nature of train station operations.

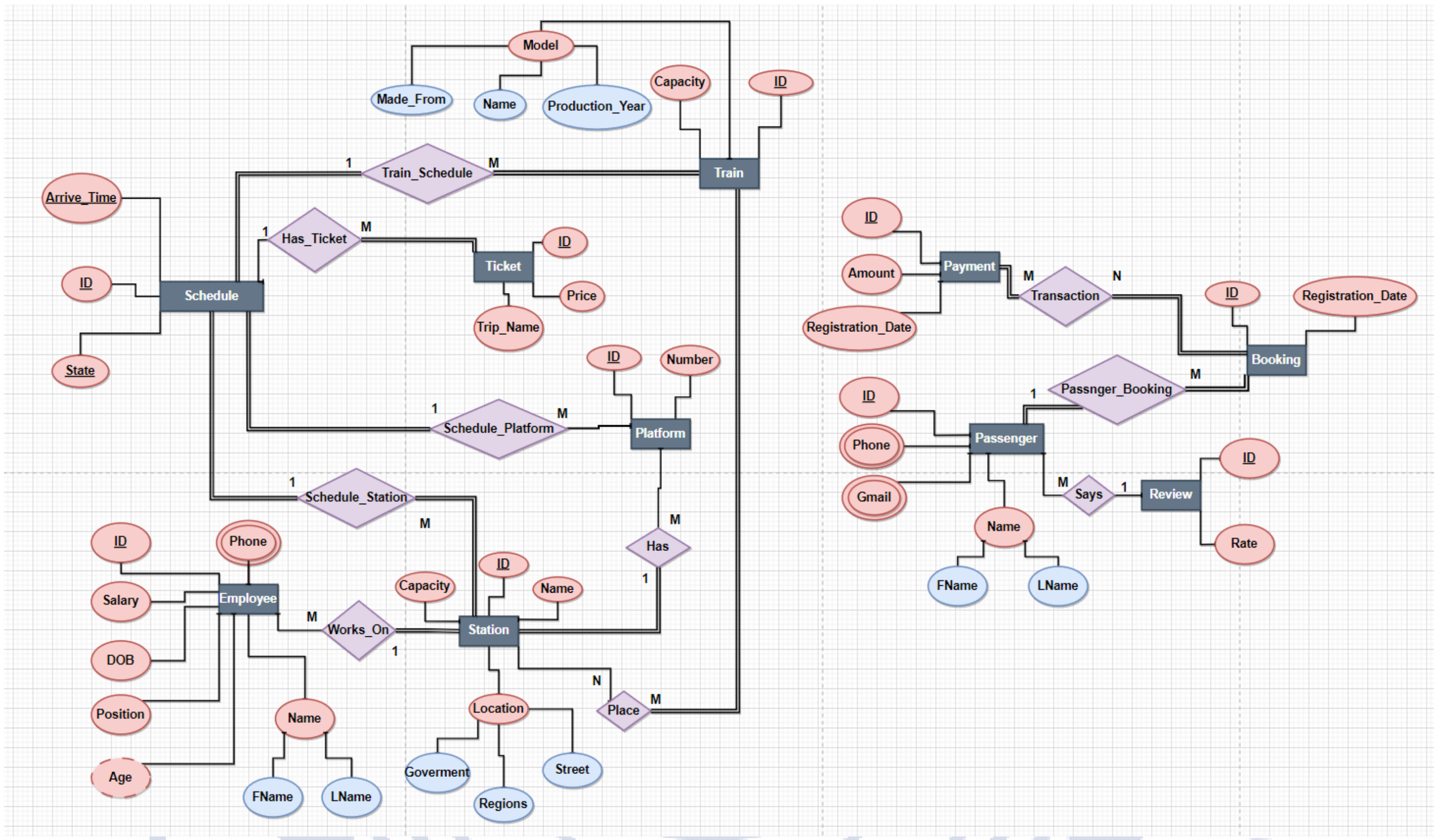
As a result, the Train Station System database emerges not only as a repository of information but as a strategic tool for decision-making, operational efficiency, and customer satisfaction. The robust data integrity mechanisms embedded within the schema contribute to the reliability of information, fostering a single source of truth for the entire train station system. This, in turn, positions the database as an invaluable asset for streamlining operations, optimizing resource utilization, and enhancing the overall experience for both passengers and staff.

In navigating the intricate landscape of train station management, the database serves as a central hub, orchestrating the seamless coordination of diverse entities. Whether it's the relationship between stations and platforms, the dynamic workforce management facilitated by employee entities, or the intricate details of scheduling and ticketing, the Train Station System database exemplifies a comprehensive and effective solution.

In essence, the success of the Train Station System hinges on the meticulous design and implementation of its database schema. Through prioritizing data integrity, the system not only meets the demands of the present but also lays a robust foundation for future scalability and innovation in the realm of train station management.

7.Entity-Relationship Diagram (ERD)

The ERD visually represents the database schema and relationships.



For open ERD model

Ahmed Mohamed
&omar khaled_.drawio

8. SQL Creation & Insertion

Creation Query:

```
CREATE TABLE Station (
    ID INT PRIMARY KEY,
    Name VARCHAR(50),
    Capacity INT,
    Government VARCHAR(50),
    Region VARCHAR(50),
    Street VARCHAR(100),
    STATION_ID INT,
    FOREIGN KEY(STATION_ID) REFERENCES Station(ID) ON DELETE CASCADE
);
--------------------------------
CREATE TABLE Platform (
    ID INT PRIMARY KEY,
    Number INT,
    Station_id INT,
    Schedule_id INT,
    FOREIGN KEY (Station_id) REFERENCES Station(ID) ON DELETE CASCADE,
    FOREIGN KEY (Schedule_id) REFERENCES `Schedule`(ID) ON DELETE CASCADE
);
------------------------------
```

```
CREATE TABLE Employee (
    ID INT PRIMARY KEY,
    FName VARCHAR(50),
    LName VARCHAR(50),
    Salary DECIMAL(10, 2),
    Position VARCHAR(50),
    DOB DATE,
    Station_ID INT,
    FOREIGN KEY (Station_ID) REFERENCES Station(ID) ON DELETE CASCADE
);
----------------------------------
CREATE TABLE Emp_phone (
    ID INT PRIMARY KEY,
    Phone INT
)
-------------------------------
CREATE TABLE Schedule (
    ID INT,
    State VARCHAR(4),
    ArriveTime TIME,
    PRIMARY KEY (ID, State, ArriveTime)
);
------------------------------------------------------
CREATE TABLE Ticket (
    ID INT PRIMARY KEY,
    Price DECIMAL(10, 2),
    Trip_Name VARCHAR(50),
    Schedule_ID INT,
    FOREIGN KEY (Schedule_ID) REFERENCES Schedule(ID) ON DELETE CASCADE
);
```

```
----------------------------------------------------
CREATE TABLE Place (
   STATION_ID INT,
   TRAIN_ID INT,
   PRIMARY KEY (STATION_ID, TRAIN_ID),
   FOREIGN KEY(STATION_ID) REFERENCES Station(ID) ON DELETE CASCADE,
   FOREIGN KEY(TRAIN_ID) REFERENCES Train(ID) ON DELETE CASCADE
);
----------------------------------------------------
CREATE TABLE Train (
   ID INT PRIMARY KEY,
   Name VARCHAR(50),
   Capacity INT,
   Made_From VARCHAR(50),
   Production_Year INT,
   Schedule_ID INT,
   FOREIGN KEY (Schedule_ID) REFERENCES Schedule(ID) ON DELETE CASCADE
);
--------------------------------------------------
CREATE TABLE Review(
     ID INT PRIMARY KEY,
   Rate INT
);
--------------------------------------------------
CREATE TABLE Passenger (
   ID INT PRIMARY KEY,
   FName VARCHAR(50),
   LName VARCHAR(50),
   Review_ID INT,
   FOREIGN KEY (Review_ID) REFERENCES Review(ID) ON DELETE CASCADE
);
------
CREATE TABLE Pass_Contact (
   ID INT PRIMARY KEY,
   Phone INT,
   Gmail VARCHAR(50)
);
------------------------------------------
CREATE TABLE Payment (
   ID INT PRIMARY KEY,
   Amount DECIMAL(10, 2),
   Registration_Date DATE
);
------------------------------------------
CREATE TABLE Booking (
   ID INT PRIMARY KEY,
   Registration_Date DATE,
   Passenger_ID INT,
   FOREIGN KEY (Passenger_ID) REFERENCES Passenger(ID) ON DELETE CASCADE
);
--------------------------------------------
CREATE TABLE `Transaction` (
     Payment_ID INT,
   Booking_ID INT,
   PRIMARY KEY (Payment_ID, Booking_ID),
   FOREIGN KEY(Payment_ID) REFERENCES Payment(ID) ON DELETE CASCADE,
   FOREIGN KEY(Booking_ID) REFERENCES Booking(ID) ON DELETE CASCADE
);
```

Insertion Query:

```
INSERT INTO Employee (ID, FName, LName, Salary, Position, DOB, Station_ID) VALUES
    (1, 'John', 'Doe', 50000, 'Conductor', '1985-04-12', 1),
    (2, 'Jane', 'Smith', 55000, 'Engineer', '1990-08-25', 2),
    (3, 'Alice', 'Brown', 45000, 'Ticket Inspector', '1982-11-16', 3),
    (4, 'Bob', 'Jones', 60000, 'Manager', '1979-02-03', 4),
    (5, 'Carol', 'White', 65000, 'Maintenance', '1995-07-30', 5);
*********************************************
INSERT INTO Emp_phone (ID,Phone) VALUES
    (1,321478965),
    (2,987412365),
    (3,854679124),
    (4,357951486),
    (5,200148956);
*********************************************
INSERT INTO Station (ID, Name, Capacity, Government, Region, Street, STATION_ID) VALUES
    (1, 'Central Station', 10, 'Government1', 'Region1', 'Street1', 1),
    (2, 'West Station', 8, 'Government2', 'Region2', 'Street2', 2),
    (3, 'East Station', 7, 'Government3', 'Region3', 'Street3', 3),
    (4, 'North Station', 5, 'Government4', 'Region4', 'Street4', 4),
    (5, 'South Station', 9, 'Government5', 'Region5', 'Street5', 5);
**********************************************
INSERT INTO Platform (ID, Number, Station_id, Schedule_id) VALUES
(1, 1, 1, 1),
(2, 2, 2, 2),
(3, 3, 3, 3),
(4, 4, 4, 4),
(5, 2, 5, 5);
**********************************************
```

```
INSERT INTO Train (ID, Name, Capacity, Made_From, Production_Year, Schedule_ID) VALUES
   (1, 'Lightning', 300, 'Steel', 2010, 1),
   (2, 'Thunder', 200, 'Aluminum', 2012, 2),
   (3, 'Storm', 250, 'Steel', 2015, 3),
   (4, 'Tornado', 350, 'Composite', 2018, 4),
   (5, 'Hurricane', 500, 'Steel', 2020, 5);
************************************************
INSERT INTO `Schedule` (ID, State, ArriveTime) VALUES
(1, 'GO', '08:00:00'),
(2, 'BACK', '09:15:00'),
(3, 'BACK', '10:30:00'),
(4, 'GO', '11:45:00'),
(5, 'GO', '13:00:00');
************************************************
INSERT INTO Ticket (ID, Price, Trip_Name, Schedule_ID) VALUES
(1, 25.50, 'Morning Express', 1),
(2, 15.75, 'Afternoon Local', 2),
(3, 40.00, 'Evening Regional', 3),
(4, 30.25, 'Night InterCity', 4),
(5, 20.00, 'Early Bird Freight', 5);
*************************************************
INSERT INTO Review (ID, Rate) VALUES
(1, 2),
(2, 7),
(3, 8),
(4, 4),
(5, 10);
*************************************************
```

```
INSERT INTO Passenger (ID, FName, LName, Review_ID) VALUES
    (1, 'John', 'Doe', 1),
    (2, 'Jane', 'Smith', 2),
    (3, 'Alice', 'Brown', 3),
    (4, 'Bob', 'Jones', 4),
    (5, 'Carol', 'White', 5);
************************************************
INSERT INTO Payment (ID, Amount, Registration_Date) VALUES
(1, 100.00, '2023-12-01'),
(2, 250.30, '2023-12-02'),
(3, 140.00, '2023-12-03'),
(4, 230.10, '2023-12-04'),
(5, 84.60, '2023-12-05');
**********************************************
INSERT INTO Booking (ID, Registration_Date, Passenger_ID) VALUES
(1, '2023-12-02', 1),
(2, '2023-12-05', 2),
(3, '2023-12-03', 3),
(4, '2023-12-08', 4),
(5, '2023-12-12', 5);
**********************************************
INSERT INTO `Pass_Contact`(ID, Phone, Gmail) VALUES
(1, 789456157, 'rat34@gmail.com'),
(2, 589641251, 'cat24@gmail.com'),
(3, 448962769, 'light2loop41@gmail.com'),
(4, 365477412, 'mail55sam34@gmail.com'),
(5, 400832170, 'warnotme@gmail.com');
**********************************************
```

```
INSERT INTO `Transaction`(Payment_ID, Booking_ID) VALUES
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(5, 5);
**********************************************
INSERT INTO Place(STATION_ID, TRAIN_ID) VALUES
(1, 1),
(2, 2),
(3, 3),
(4, 4),
(5, 5);
```