1. Introduction

The railway reservation system is an essential part of the transportation sector since it promotes resource efficiency and assists in controlling the flow of passengers. However, there are numerous issues with the present manual reservation system, including lengthy wait times, ticket shortages, and a lack of transparency in the booking procedure. We suggest a project to manage a railway reservation database with the goal of automating the reservation procedure and offering passengers a smooth experience in order to address these problems. The latest technology and database management best practices will be included into a centralized database system that we are designing and developing. The goal of this project is to develop an effective and dependable reservation system that will promote overall customer satisfaction, expedite business processes, and eventually improve the railway company's reputation.

2. Entities and Relationships

Entities:

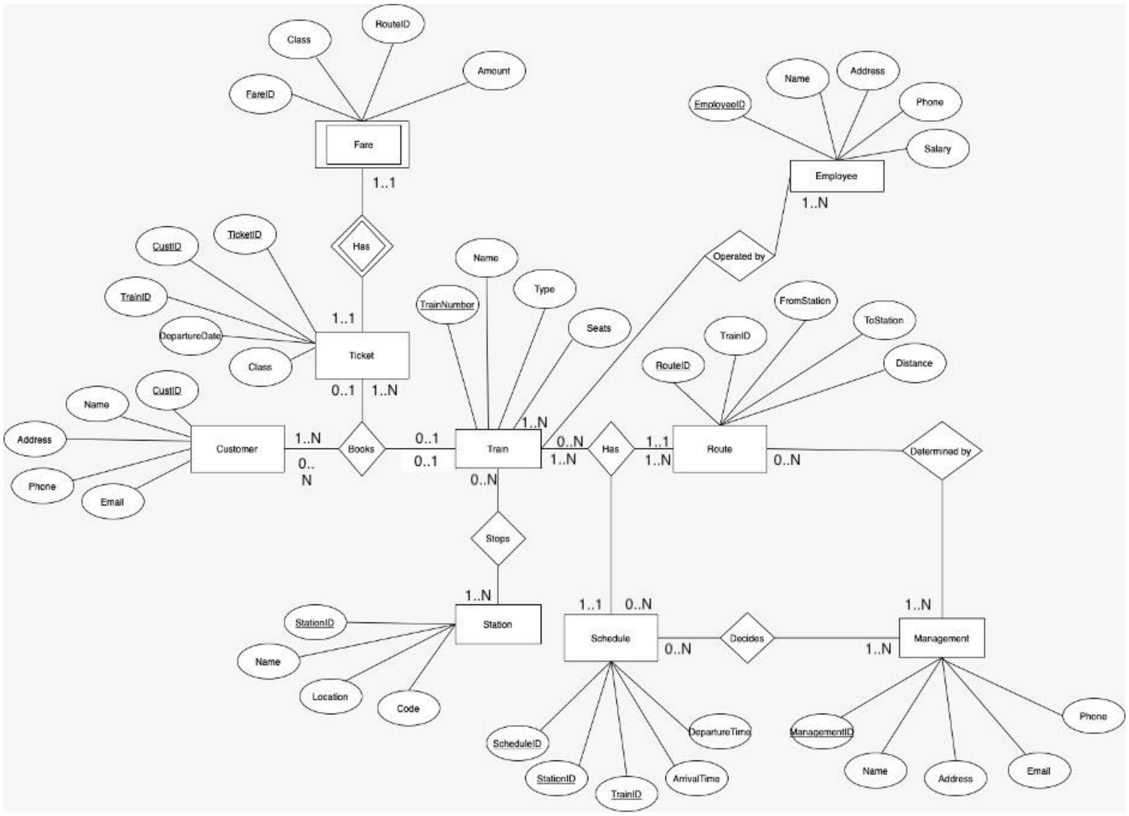
1. Customer: a person who books a ticket for railway transportation.
2. Train: a vehicle that provides railway transportation services.
3. Ticket: a document that grants a customer the right to travel on a specific train.
4. Station: a place where trains stop to pick up and drop off passengers.
5. Schedule: a plan that defines the departure and arrival times of trains at each station.
6. Route: a path that a train takes from one station to another.
7. Fare: the amount of money a customer pays for a ticket.
8. Employee: a person who works for the railway company and provides various services to customers.
9. Management: the team responsible for overseeing the overall operations of the railway company.

Relationships:

1. A customer books a ticket for a specific train.
2. A train operates on a specific route and has a defined schedule.
3. A ticket is associated with a specific train and a specific customer.
4. A train stops at multiple stations as part of its route.
5. A schedule defines the departure and arrival times of a train at each station.
6. A route is made up of multiple stations.
7. The fare for a ticket is determined by the route and class of service.
8. Employees provide various services to customers at the station or on the train.

9. Management is responsible for overseeing the operations of the railway company and ensuring that all employees are providing high-quality services to customers.

3. ERR Model



CARDINALITIES:
A customer can book multiple train (1:N)
A train can have multiple routes (1:N)
A train can have only multiple schedule (1:N) Many employees can work in multiple train (N:N)
Many customers can book one tickets (N:1) Management can decide multiple schedule (1:N)
Management determines many routes (1:N)

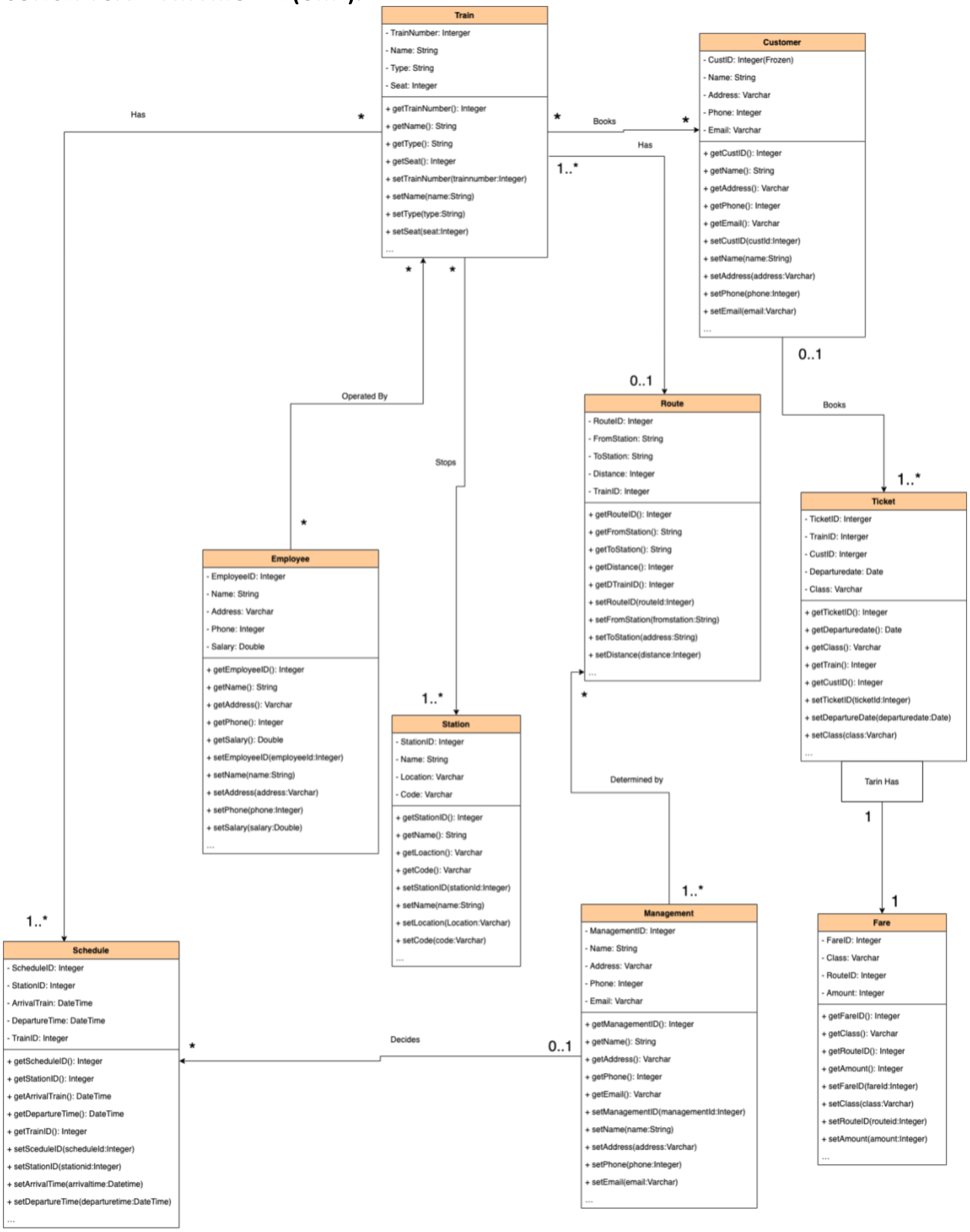
Transaction:

On February 24th, 2023, customer Niloufer purchased a train ticket for the 9:00am express train from Station Boston to Station Newyork. The fare for the ticket was $50. The train is scheduled to depart on time and follow Route 1. The ticket was issued by Employee parag at the station's ticket counter.

Reference:

Niloufer syed. Train ticket purchase for 9:00am express train from Station A to Station B on February 24th, 2023. Fare: $50. Route 1. Issued by Parag, J. at Station A ticket counter.

CONCEPTUAL DATA MODEL (UML): -



Relationship Model: -

Customer (customer_id, name, email, phone_number, address)
customer_ID is the primary key, NULL NOT ALLOWED

Ticket (Ticket_id, *trainID*, class, departureDate, *customerID*)
Ticket_id is the primary key, NULL NOT ALLOWED
Customerid is a foreign key from customer, NULL NOT ALLOWED
Trained is a foreign key from train, NULL NOT ALLOWED

Train (trainid, type, name, Seats)
Trainid is the primary key, NULL NOT ALLOWED

Route (Route_id, *TrainID*, FromStation, Tostation, Distance)
Route_ID is the primary key, NULL NOT ALLOWED
Train_ID is the foreign key from Train, NULL NOT ALLOWED

Employee (employee_id, name, address, phone, salary)
Employee_ID is a primary key, NULL NOT ALLOWED

Management (management_id, *employee_id,* name, email, phone_number, address)
Management_ID is a primary key, NULL NOT ALLOWED
employee_id is a foreign key from employee, NULL NOT ALLOWED

Station (station_id, *Route_ID,* name, location, code)
Station_ID is a primary key, NULL NOT ALLOWED
Route_id is a foreign key from route, NULL NOT ALLOWED

Fare (*fare_id*, *TicketID,* class)
Fare_ID is a foreign key, NULL NOT ALLOWED
Ticket_ID is a foreign key from Ticket, NOT NULL ALLOWED

Schedule (schedule_id, *Management_id, Train_id*, departure_time, arrival_time)
Schedule_ID is a primary key, NULL NOT ALLOWED
Train_id is a foreign key from train, NULL NOT ALLOWED
Management_id is a foreign key from management, NULL NOT ALLOWED

1NF (First Normal Form): The data is in 1NF as there are no repeating groups, and each table has a primary key.

2NF (Second Normal Form): In the Ticket table, class and departureDate columns are dependent on the trainID column, which is part of the primary key. We can split the Ticket table into two tables:

Customer (customer_id, name, email, phone_number, address)

customer_id is the primary key, NULL NOT ALLOWED

Ticket (Ticket_id, trainID, *customerID*)

Ticket_id is the primary key, NULL NOT ALLOWED

customerID is a foreign key from customer, NULL NOT ALLOWED

Train (trainid, type, name, Seats)

trainid is the primary key, NULL NOT ALLOWED

Ticket_Details (*Ticket_id*, class, departureDate)

Ticket_id is a foreign key from Ticket, NOT NULL ALLOWED

3NF (Third Normal Form): In the Route table, FromStation and ToStation columns are dependent on the Route_ID column, which is the primary key. We can split the Route table into two tables:

Route (Route_id, *TrainID*, Distance)

Route_id is the primary key, NULL NOT ALLOWED

TrainID is a foreign key from Train, NULL NOT ALLOWED

Station (station_id, *Route_ID*, name, location, code)

station_id is the primary key, NULL NOT ALLOWED

Route_ID is a foreign key from Route, NULL NOT ALLOWED

In the Schedule table, departure_time and arrival_time columns are dependent on the Train_id and Management_id columns, which are both part of the primary key. We can split the Schedule table into two tables:

Schedule (schedule_id, *Train_id*, *Management_id*)

schedule_id is the primary key, NULL NOT ALLOWED

Train_id is a foreign key from Train, NULL NOT ALLOWED

Management_id is a foreign key from Management, NULL NOT ALLOWED

Schedule_Details (*schedule_id*, departure_time, arrival_time)

schedule_id is a foreign key from Schedule, NOT NULL ALLOWED

3.5NF (Third and a Half Normal Form): In the Management table, name, email, phone_number, and address columns are dependent on the management_id column, which is the primary key. We can split the Management table into two tables:

Management (management_id, *employee_id*)

management_id is the primary key, NULL NOT ALLOWED

employee_id is a foreign key from Employee, NULL NOT ALLOWED

Management_Details (*management_id*, name, email, phone_number, address)

management_id is a foreign key from Management, NOT NULL ALLOWED

In the Fare table, class column is dependent on the TicketID column, which is part of the primary key. We can split the Fare table into two tables:

Fare (fare_id, *TicketID*)

fare_id is the primary key, NULL NOT ALLOWED

TicketID is a foreign key from Ticket, NOT NULL ALLOWED

Fare_Details (*fare_id*, class)

fare_id is a foreign key from Fare, NOT NULL ALLOWED

This would result in the following normalized tables:

Customer (customer_id, name, email, phone_number, address)

Ticket (Ticket_id, trainID, customerID)

Train (trainid, type, name, Seats)

Ticket_Details (Ticket_id, class, departureDate)

Route (Route_id, TrainID, Distance)

Station (station_id, Route_ID, name, location, code)

Schedule (schedule_id, Train_id, Management_id)

Schedule_Details (schedule_id, departure_time, arrival_time)

Employee (employee_id, name, address, phone, salary)

Management (management_id, employee_id)

Management_Details (management_id, name, email, phone_number, address)

Fare (fare_id, *TicketID*)

Fare_Details (*fare_id*, class)

Model Implementation

- **Creating empty tables for the future analysis**

Tables created are:

1. Customer
2. Train
3. Route
4. Employee
5. Management
6. Station
7. Ticket
8. Fare
9. Schedule

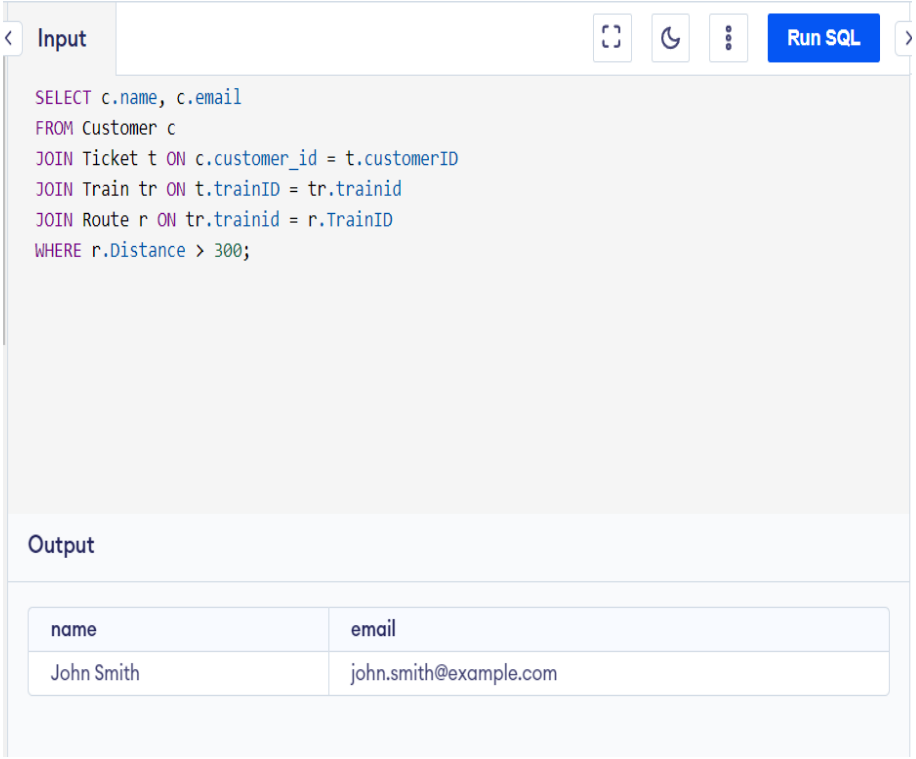
Testing the tables with queries on the sample data

Q1) Display the name and email address of all customers who have purchased a ticket for a train with a distance greater than 300 km.

Query

```
SELECT c.name, c.email
FROM Customer c
JOIN Ticket t ON c.customer_id = t.customerID
JOIN Train tr ON t.trainID = tr.trainid
JOIN Route r ON tr.trainid = r.TrainID
WHERE r.Distance > 300;
```

Output:



```
SELECT c.name, c.email
FROM Customer c
JOIN Ticket t ON c.customer_id = t.customerID
JOIN Train tr ON t.trainID = tr.trainid
JOIN Route r ON tr.trainid = r.TrainID
WHERE r.Distance > 300;
```

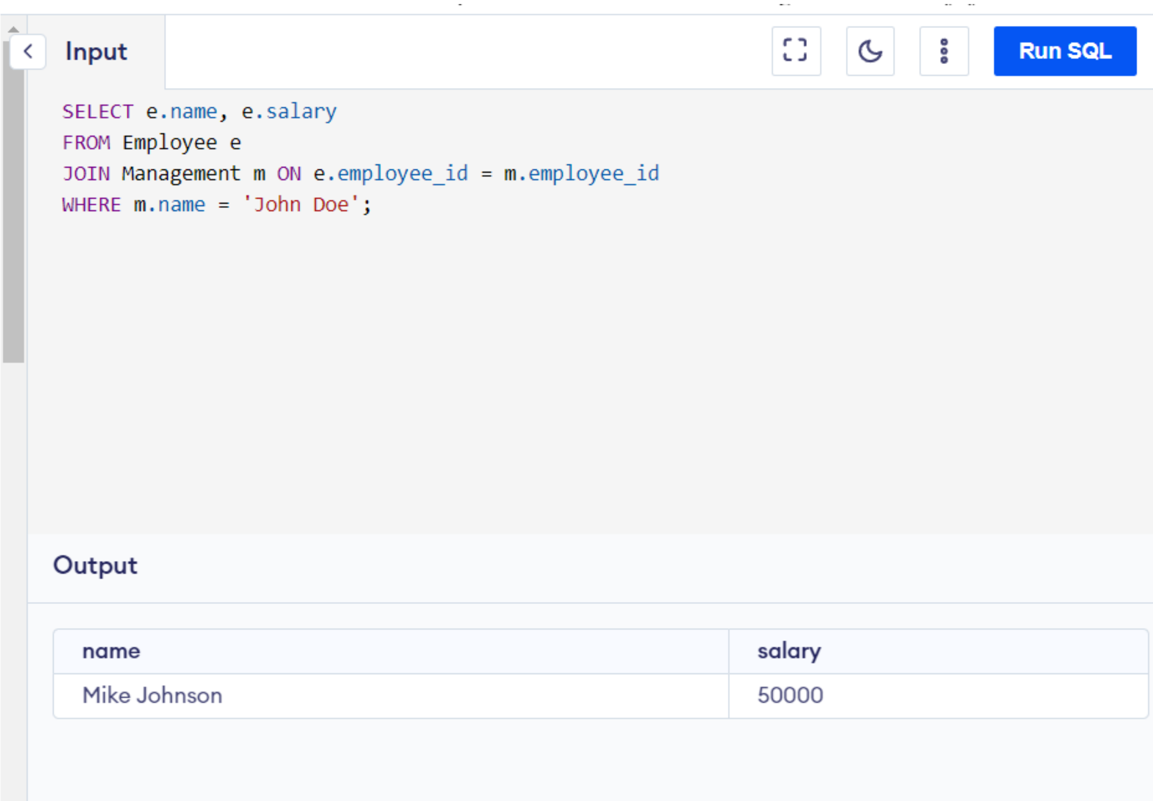
name	email
John Smith	john.smith@example.com

Q2) Display the name and salary of all employees who are managed by John Doe.

Query

```
SELECT e.name, e.salary
FROM Employee e
JOIN Management m ON e.employee_id = m.employee_id
WHERE m.name = 'John Doe';
```

Output



```
SELECT e.name, e.salary
FROM Employee e
JOIN Management m ON e.employee_id = m.employee_id
WHERE m.name = 'John Doe';
```

Output

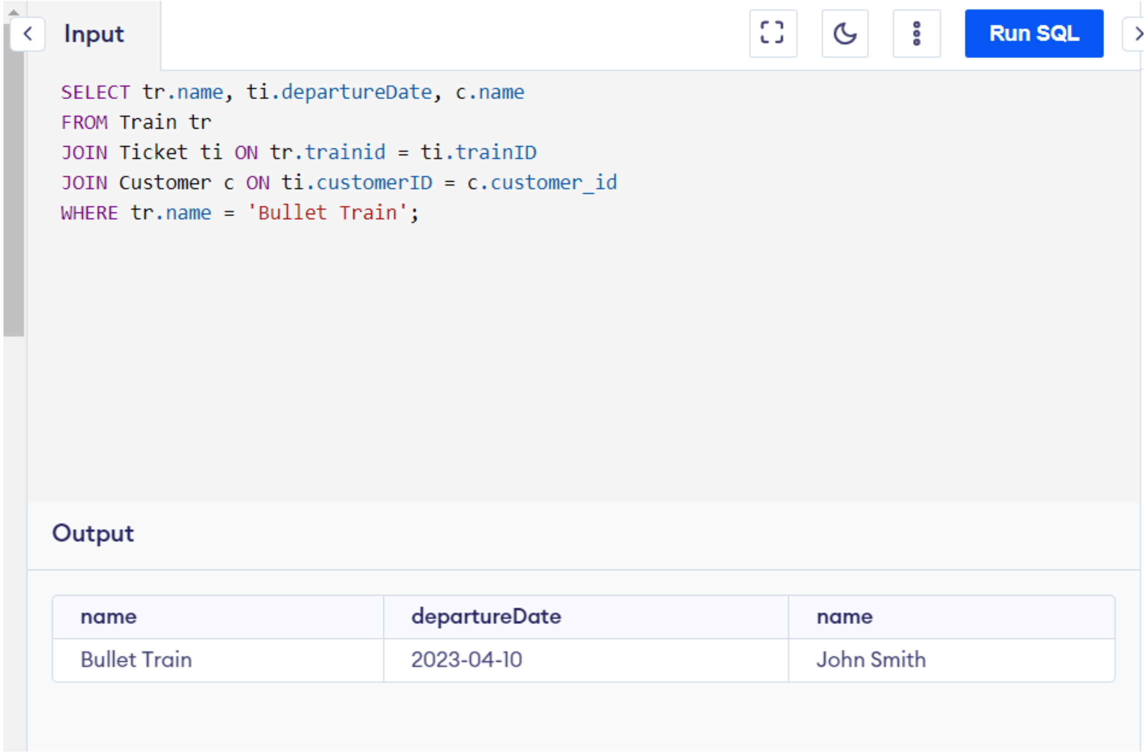
name	salary
Mike Johnson	50000

Q3) Display the name of the train, departure date, and customer name for all tickets sold for the train named "Bullet Train".

Query

```
SELECT tr.name, ti.departureDate, c.name
FROM Train cpscoop@northeastern.edutr
JOIN Ticket ti ON tr.trainid = ti.trainID
JOIN Customer c ON ti.customerID = c.customer_id
WHERE tr.name = 'Bullet Train';
```

Output



```
SELECT tr.name, ti.departureDate, c.name
FROM Train tr
JOIN Ticket ti ON tr.trainid = ti.trainID
JOIN Customer c ON ti.customerID = c.customer_id
WHERE tr.name = 'Bullet Train';
```

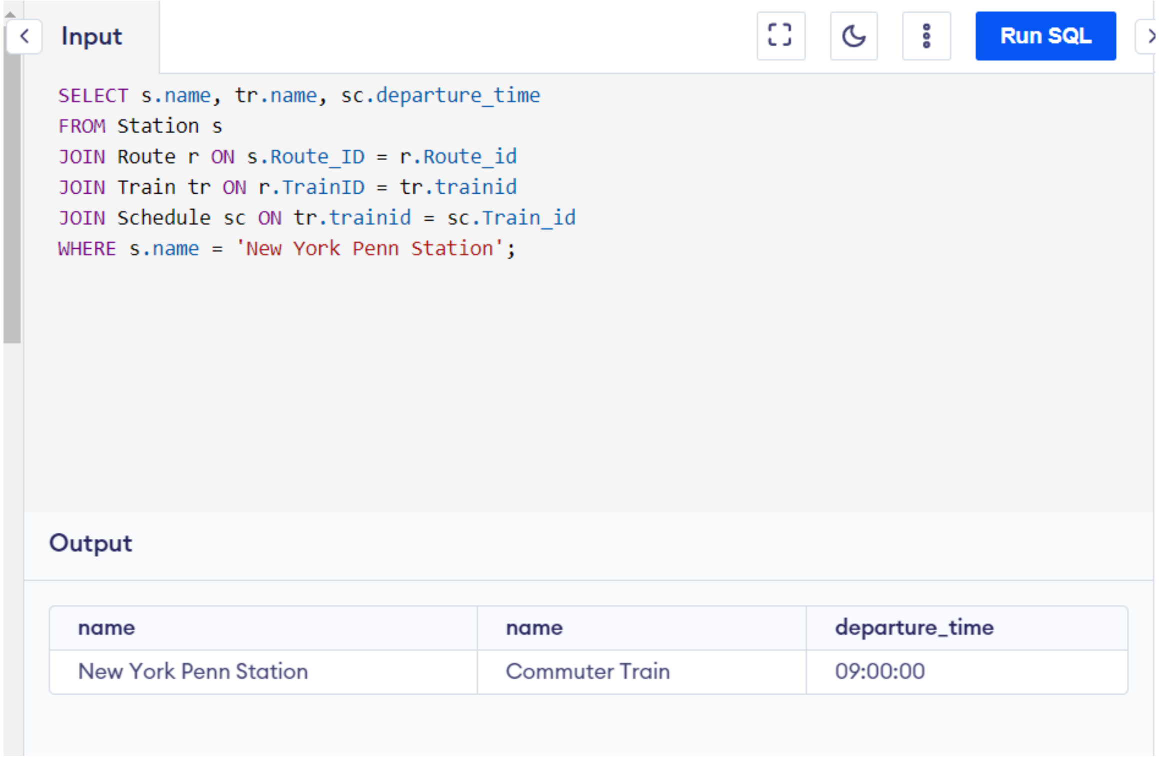
name	departureDate	name
Bullet Train	2023-04-10	John Smith

Q4)Display the name of the station, train name, and departure time for all schedules that depart from New York Penn Station.

Query

```
SELECT s.name, tr.name, sc.departure_time
FROM Station s
JOIN Route r ON s.Route_ID = r.Route_id
JOIN Train tr ON r.TrainID = tr.trainid
JOIN Schedule sc ON tr.trainid = sc.Train_id
WHERE s.name = 'New York Penn Station';
```

Output



```
SELECT s.name, tr.name, sc.departure_time
FROM Station s
JOIN Route r ON s.Route_ID = r.Route_id
JOIN Train tr ON r.TrainID = tr.trainid
JOIN Schedule sc ON tr.trainid = sc.Train_id
WHERE s.name = 'New York Penn Station';
```

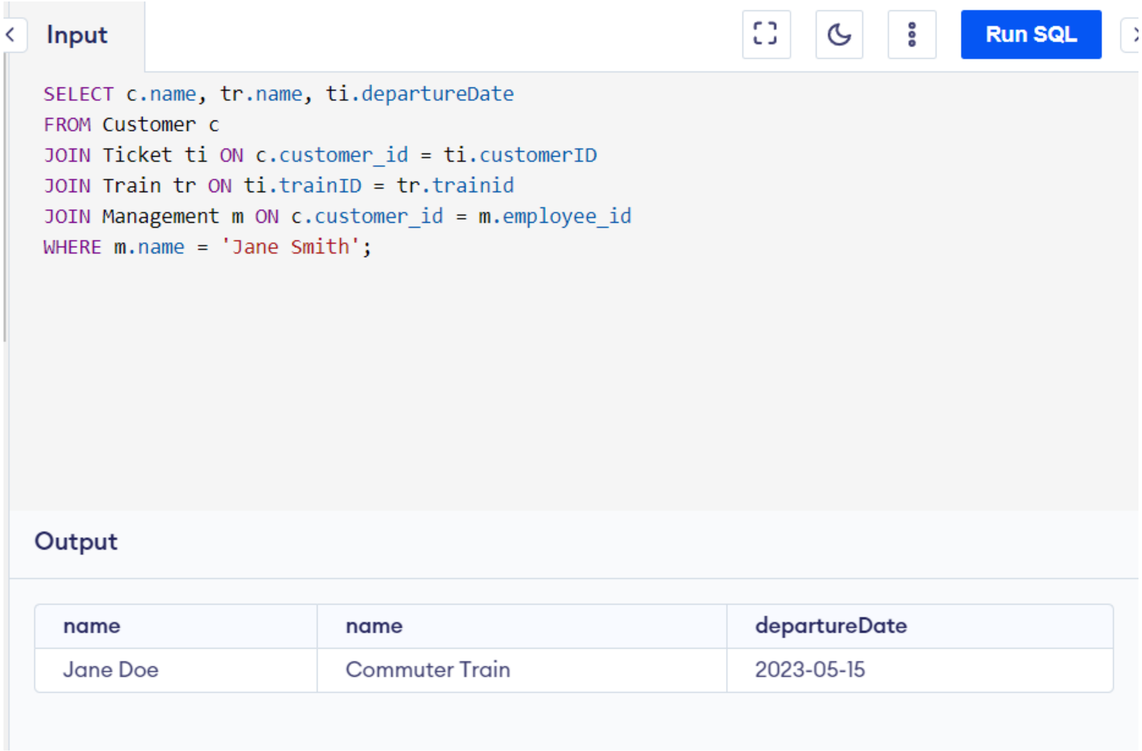
Output

name	name	departure_time
New York Penn Station	Commuter Train	09:00:00

Q5) Display the name of the customer, train name, and departure date for all tickets sold by Jane Smith.

Query

```
SELECT c.name, tr.name, ti.departureDate
FROM Customer c
JOIN Ticket ti ON c.customer_id = ti.customerID
JOIN Train tr ON ti.trainID = tr.trainid
JOIN Management m ON c.customer_id = m.employee_id
WHERE m.name = 'Jane Smith';
```



```
SELECT c.name, tr.name, ti.departureDate
FROM Customer c
JOIN Ticket ti ON c.customer_id = ti.customerID
JOIN Train tr ON ti.trainID = tr.trainid
JOIN Management m ON c.customer_id = m.employee_id
WHERE m.name = 'Jane Smith';
```

Output

name	name	departureDate
Jane Doe	Commuter Train	2023-05-15

#### Analytical Queries ############

#1. Retrieve the total number of customers who have booked tickets for a particular train:

```
SELECT COUNT(DISTINCT Ticket.customerID) AS Total_Customers FROM Ticket WHERE Ticket.trainID = 1;
```

#2. Retrieve the total number of employees who are working in a particular department:

```
SELECT COUNT(*) AS Total_Employees
FROM Employee WHERE Employee.employee_id
IN (SELECT Management.employee_id
FROM Management WHERE Management.name = 'Sales');
```

#3. Retrieve the name and type of all the trains that have more than 500 seats:

```
SELECT name, type FROM Train WHERE Seats > 500;
```

#4. Retrieve the total number of tickets booked for a particular train on a specific date:

```
SELECT COUNT(*) AS Total_Tickets FROM Ticket WHERE Ticket.trainID = 2 AND Ticket.departureDate =
'2023-04-12';
```

#5. Retrieve the name and type of all the trains that have stations at a particular location(NEW YORK):

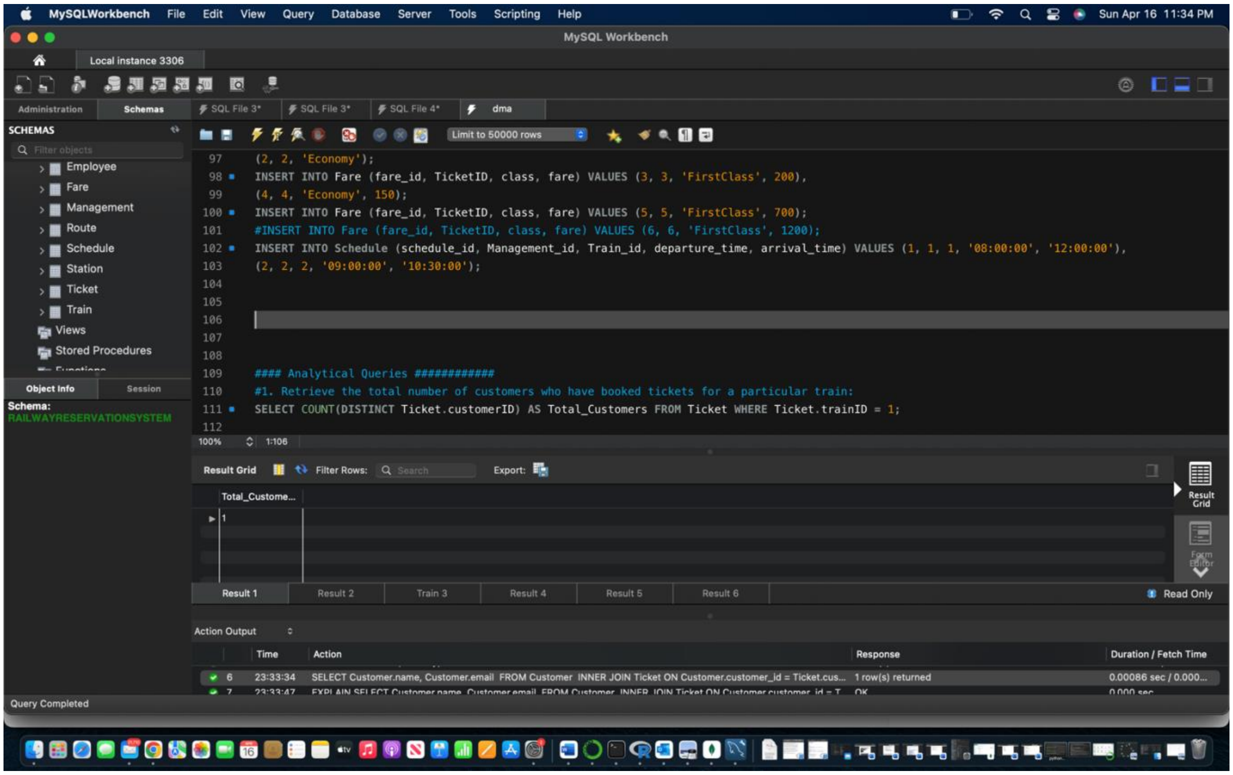
```
SELECT Train.name, Train.type
FROM Train
INNER JOIN Route ON Train.trainid = Route.TrainID
INNER JOIN Station ON Route.Route_id = Station.Route_ID
```

WHERE Station.location = 'New York';

#6. Retrieve the name and email of the customer who has booked the maximum number of tickets:

SELECT Customer.name, Customer.email
FROM Customer
INNER JOIN Ticket ON Customer.customer_id = Ticket.customerID
GROUP BY Customer.customer_id
ORDER BY COUNT(Ticket.Ticket_id) DESC
Limit 1;

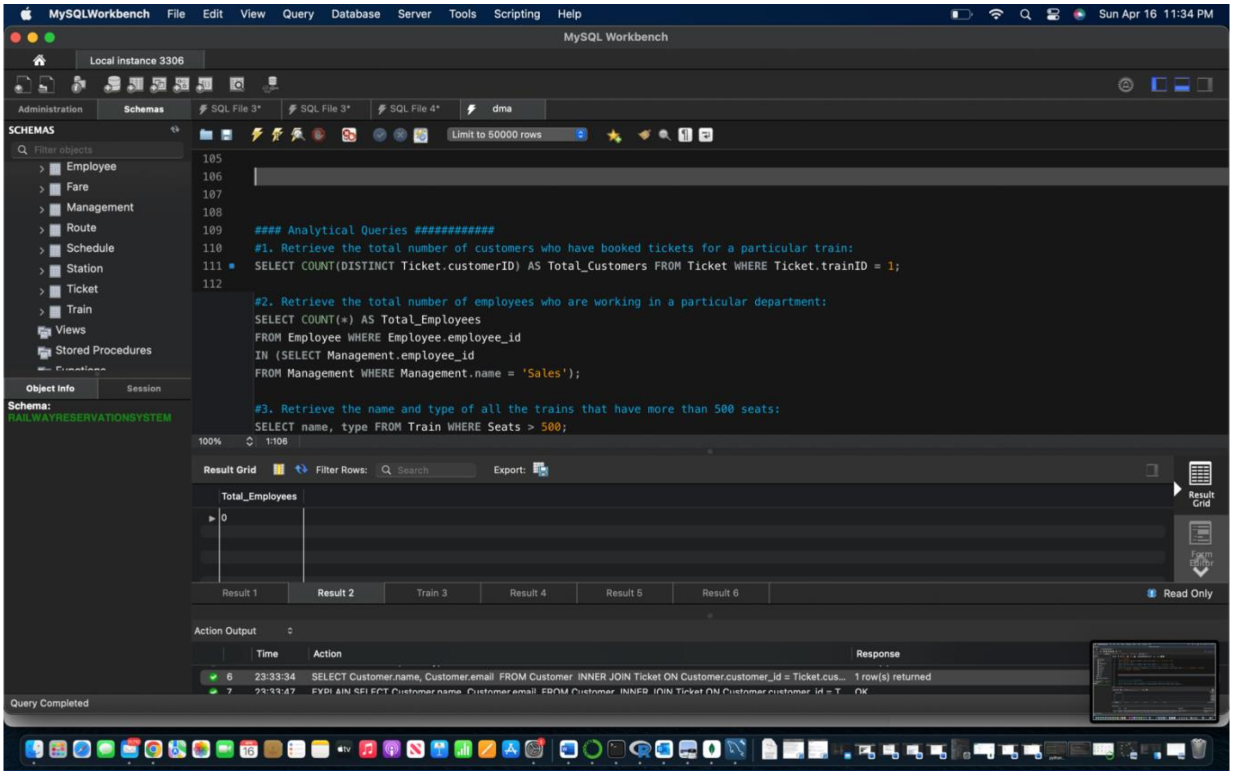
Output:



```
(2, 2, 'Economy');
INSERT INTO Fare (fare_id, TicketID, class, fare) VALUES (3, 3, 'FirstClass', 200),
(4, 4, 'Economy', 150);
INSERT INTO Fare (fare_id, TicketID, class, fare) VALUES (5, 5, 'FirstClass', 700);
#INSERT INTO Fare (fare_id, TicketID, class, fare) VALUES (6, 6, 'FirstClass', 1200);
INSERT INTO Schedule (schedule_id, Management_id, Train_id, departure_time, arrival_time) VALUES (1, 1, 1, '08:00:00', '12:00:00'),
(2, 2, 2, '09:00:00', '10:30:00');




#### Analytical Queries ############
#1. Retrieve the total number of customers who have booked tickets for a particular train:
SELECT COUNT(DISTINCT Ticket.customerID) AS Total_Customers FROM Ticket WHERE Ticket.trainID = 1;
```

Total_Custome...
1

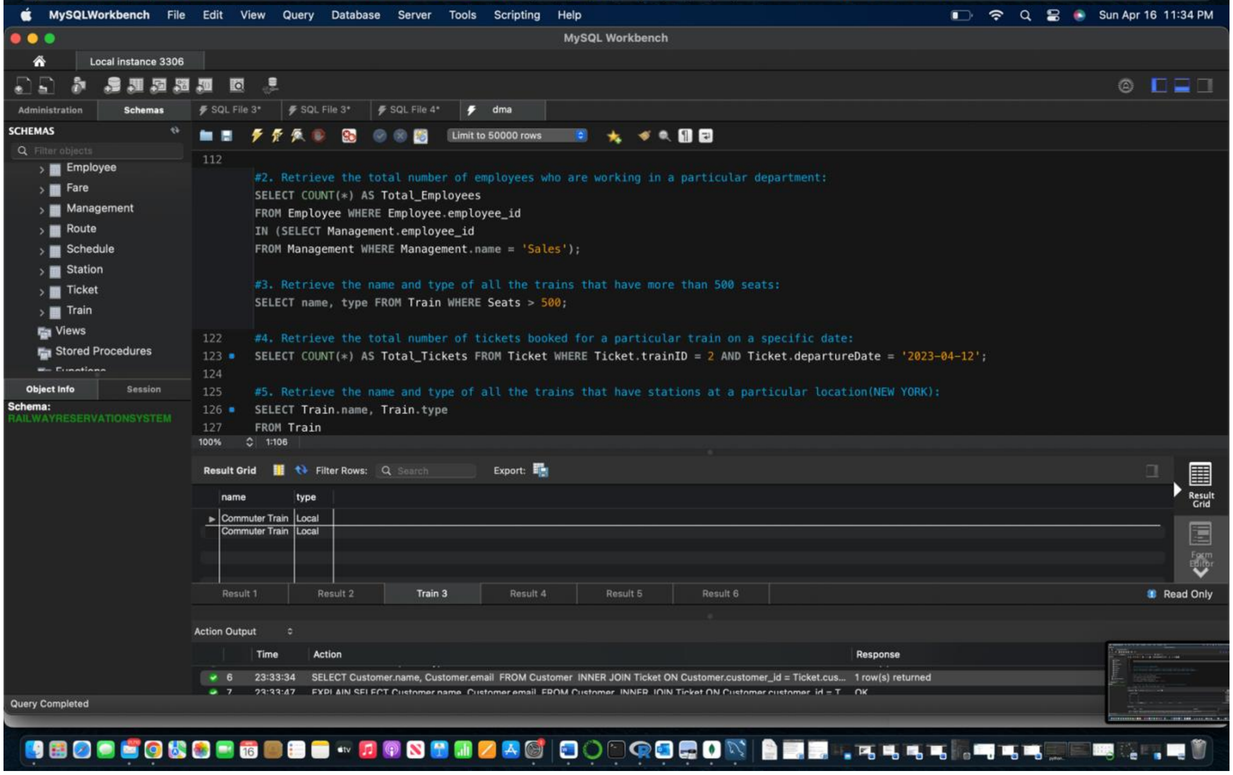
	Time	Action	Response	Duration / Fetch Time
6	23:33:34	SELECT Customer.name, Customer.email FROM Customer INNER JOIN Ticket ON Customer.customer_id = Ticket.cus...	1 row(s) returned	0.00086 sec / 0.000...
7	23:33:47	EXPLAIN SELECT Customer.name, Customer.email FROM Customer INNER JOIN Ticket ON Customer.customer_id = T...	OK	0.000 sec



```
#### Analytical Queries ############
#1. Retrieve the total number of customers who have booked tickets for a particular train:
SELECT COUNT(DISTINCT Ticket.customerID) AS Total_Customers FROM Ticket WHERE Ticket.trainID = 1;

#2. Retrieve the total number of employees who are working in a particular department:
SELECT COUNT(*) AS Total_Employees
FROM Employee WHERE Employee.employee_id
IN (SELECT Management.employee_id
FROM Management WHERE Management.name = 'Sales');

#3. Retrieve the name and type of all the trains that have more than 500 seats:
SELECT name, type FROM Train WHERE Seats > 500;
```

Total_Employees
0

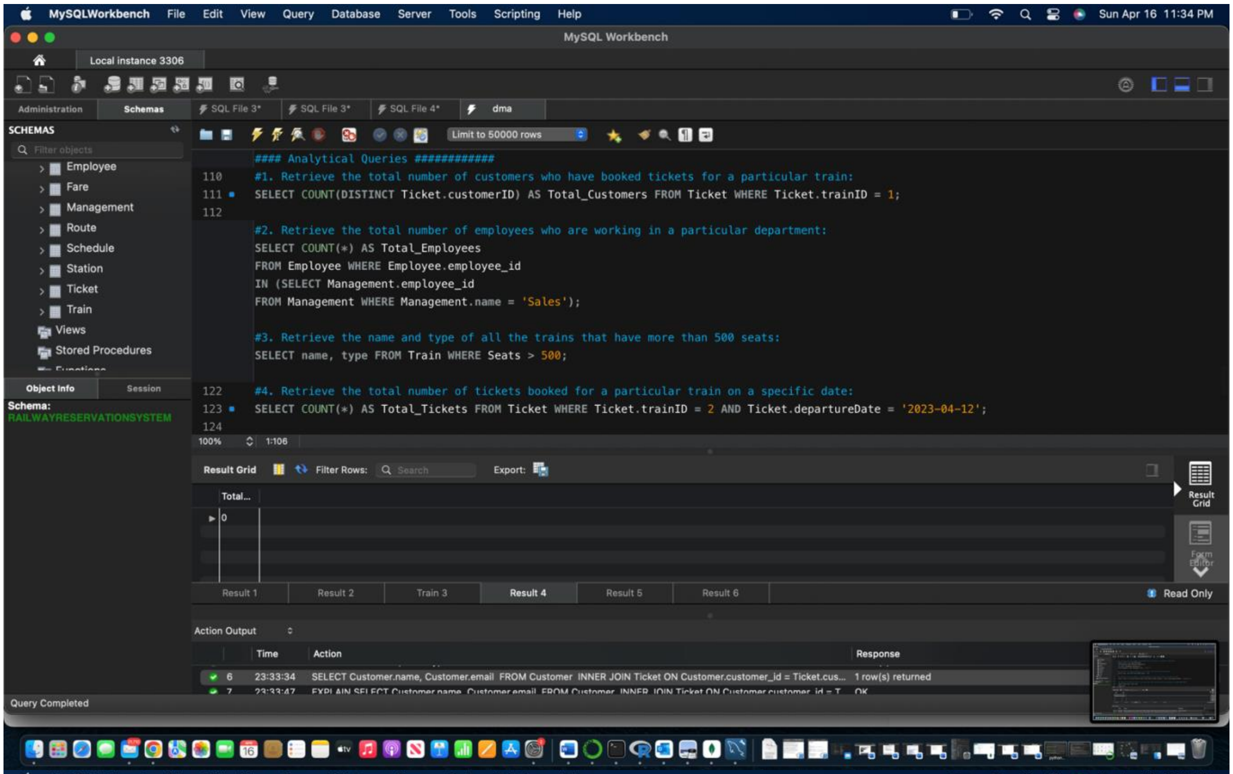
	Time	Action	Response
6	23:33:34	SELECT Customer.name, Customer.email FROM Customer INNER JOIN Ticket ON Customer.customer_id = Ticket.cus...	1 row(s) returned
7	23:33:47	EXPLAIN SELECT Customer.name, Customer.email FROM Customer INNER JOIN Ticket ON Customer.customer_id = T...	OK



```
#2. Retrieve the total number of employees who are working in a particular department:
SELECT COUNT(*) AS Total_Employees
FROM Employee WHERE Employee.employee_id
IN (SELECT Management.employee_id
FROM Management WHERE Management.name = 'Sales');

#3. Retrieve the name and type of all the trains that have more than 500 seats:
SELECT name, type FROM Train WHERE Seats > 500;

#4. Retrieve the total number of tickets booked for a particular train on a specific date:
SELECT COUNT(*) AS Total_Tickets FROM Ticket WHERE Ticket.trainID = 2 AND Ticket.departureDate = '2023-04-12';

#5. Retrieve the name and type of all the trains that have stations at a particular location(NEW YORK):
SELECT Train.name, Train.type
FROM Train
```

name	type
Commuter Train	Local
Commuter Train	Local

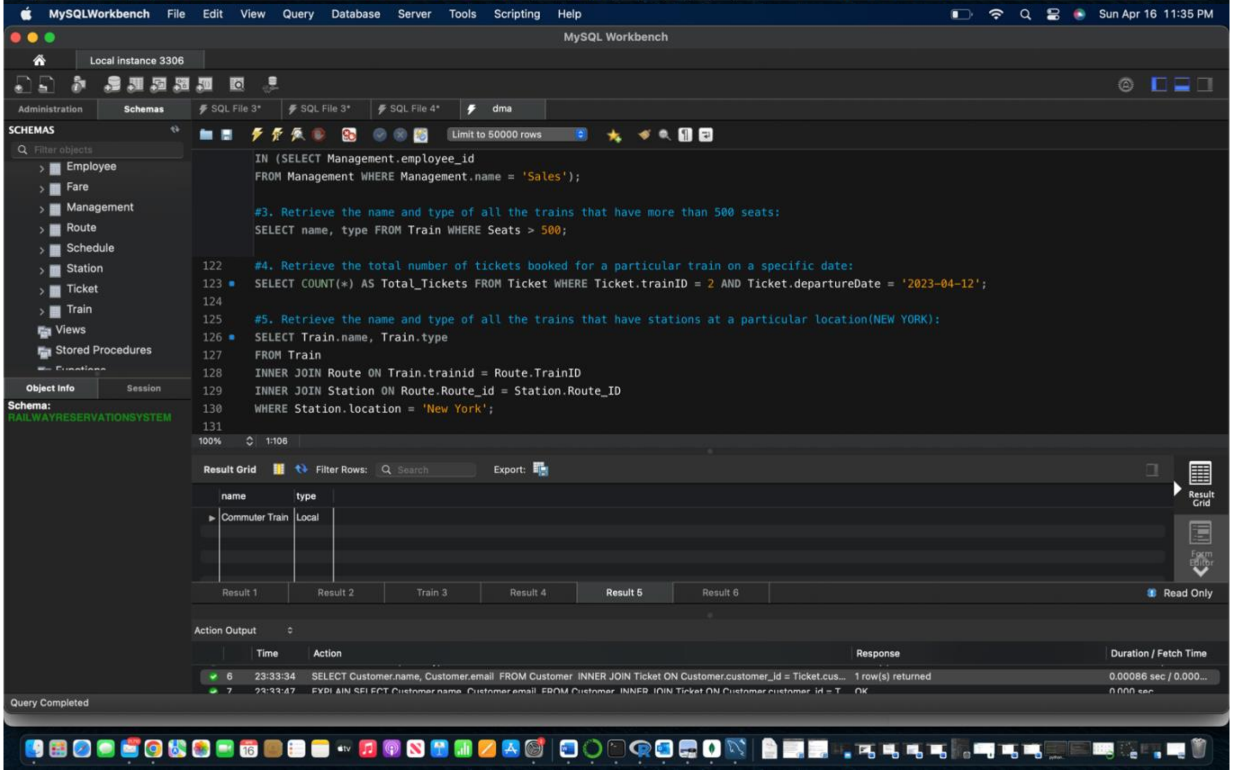
	Time	Action	Response
6	23:33:34	SELECT Customer.name, Customer.email FROM Customer INNER JOIN Ticket ON Customer.customer_id = Ticket.cus...	1 row(s) returned
7	23:33:47	EXPLAIN SELECT Customer.name, Customer.email FROM Customer INNER JOIN Ticket ON Customer.customer_id = T...	OK



```
#### Analytical Queries ############
#1. Retrieve the total number of customers who have booked tickets for a particular train:
SELECT COUNT(DISTINCT Ticket.customerID) AS Total_Customers FROM Ticket WHERE Ticket.trainID = 1;

#2. Retrieve the total number of employees who are working in a particular department:
SELECT COUNT(*) AS Total_Employees
FROM Employee WHERE Employee.employee_id
IN (SELECT Management.employee_id
FROM Management WHERE Management.name = 'Sales');

#3. Retrieve the name and type of all the trains that have more than 500 seats:
SELECT name, type FROM Train WHERE Seats > 500;

#4. Retrieve the total number of tickets booked for a particular train on a specific date:
SELECT COUNT(*) AS Total_Tickets FROM Ticket WHERE Ticket.trainID = 2 AND Ticket.departureDate = '2023-04-12';
```

Total...
0

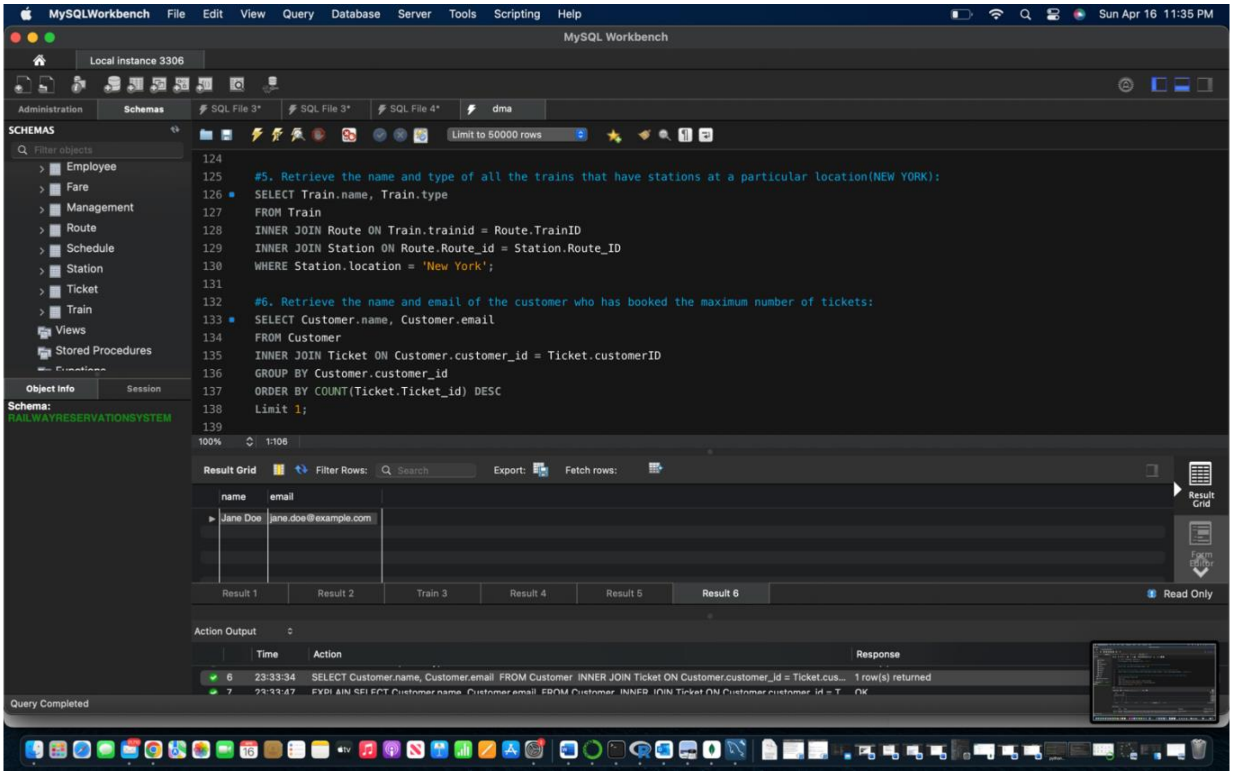
```
   Time      Action                                                                                                Response
6  23:33:34  SELECT Customer.name, Customer.email  FROM Customer  INNER JOIN Ticket ON Customer.customer_id = Ticket.cus...  1 row(s) returned
7  23:33:47  EXPLAIN SELECT Customer.name, Customer.email  FROM Customer  INNER JOIN Ticket ON Customer.customer_id = T...   OK
```



```
IN (SELECT Management.employee_id
FROM Management WHERE Management.name = 'Sales');

#3. Retrieve the name and type of all the trains that have more than 500 seats:
SELECT name, type FROM Train WHERE Seats > 500;

#4. Retrieve the total number of tickets booked for a particular train on a specific date:
SELECT COUNT(*) AS Total_Tickets FROM Ticket WHERE Ticket.trainID = 2 AND Ticket.departureDate = '2023-04-12';

#5. Retrieve the name and type of all the trains that have stations at a particular location(NEW YORK):
SELECT Train.name, Train.type
FROM Train
INNER JOIN Route ON Train.trainid = Route.TrainID
INNER JOIN Station ON Route.Route_id = Station.Route_ID
WHERE Station.location = 'New York';
```

name	type
Commuter Train	Local

```
   Time      Action                                                                                                Response           Duration / Fetch Time
6  23:33:34  SELECT Customer.name, Customer.email  FROM Customer  INNER JOIN Ticket ON Customer.customer_id = Ticket.cus...  1 row(s) returned  0.00086 sec / 0.000...
7  23:33:47  EXPLAIN SELECT Customer.name, Customer.email  FROM Customer  INNER JOIN Ticket ON Customer.customer_id = T...   OK                 0.000 sec
```



```
#5. Retrieve the name and type of all the trains that have stations at a particular location(NEW YORK):
SELECT Train.name, Train.type
FROM Train
INNER JOIN Route ON Train.trainid = Route.TrainID
INNER JOIN Station ON Route.Route_id = Station.Route_ID
WHERE Station.location = 'New York';

#6. Retrieve the name and email of the customer who has booked the maximum number of tickets:
SELECT Customer.name, Customer.email
FROM Customer
INNER JOIN Ticket ON Customer.customer_id = Ticket.customerID
GROUP BY Customer.customer_id
ORDER BY COUNT(Ticket.Ticket_id) DESC
Limit 1;
```

name	email
Jane Doe	jane.doe@example.com

```
   Time      Action                                                                                   Response
6  23:33:34  SELECT Customer.name, Customer.email  FROM Customer  INNER JOIN Ticket ON Customer.customer_id = Ticket.cus...  1 row(s) returned
7  23:33:47  EXPLAIN SELECT Customer.name, Customer.email  FROM Customer  INNER JOIN Ticket ON Customer.customer_id = T   OK
```

Python Dashboard and NOSQL Queries:

In [1]:

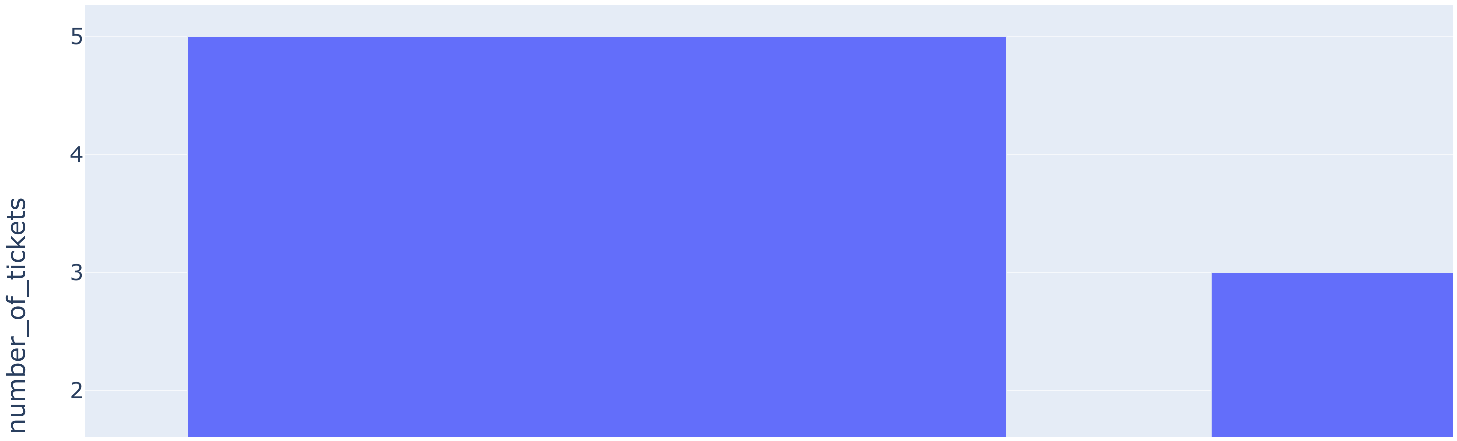
```python
import mysql.connector
import pyodbc
import pandas as pd
import plotly.express as px
from mysql.connector import Error
connection = None
try:
    connection = mysql.connector.connect(
        host='localhost',
        port='3306',
        user='root',
        password='Parag12345@',
        database='RAILWAYRESERVATIONSYSTEM',
        auth_plugin='mysql_native_password'
    )

    if connection.is_connected():
        db_Info = connection.get_server_info()
        print("Connected to MySQL Server version ", db_Info)
        cursor = connection.cursor()
        cursor.execute("select database();")
        record = cursor.fetchone()
        print("Your connected to database: ", record)
        print("\n")

        #Retrieve count of tickets for each type of train

        query1 = "SELECT Train.type, COUNT(Ticket.ticket_id) AS number_of_tickets FROM Train INNER JOIN Ticke

        #Retrieve Average fare of tickets for each type of train

        query2 = "SELECT Train.type, AVG(Fare.fare) AS average_fare FROM Train INNER JOIN Ticket ON Train.tra

        df1 = pd.read_sql(query1, connection)
        df2 = pd.read_sql(query2, connection)
        # Create visualizations using plotly
        fig1 = px.bar(df1, x='type', y='number_of_tickets', title='Number of Tickets by Train Type')
        fig2 = px.bar(df2, x='type', y='average_fare', title='Average Fare by Train Type')


        fig1.show()
        fig2.show()
```

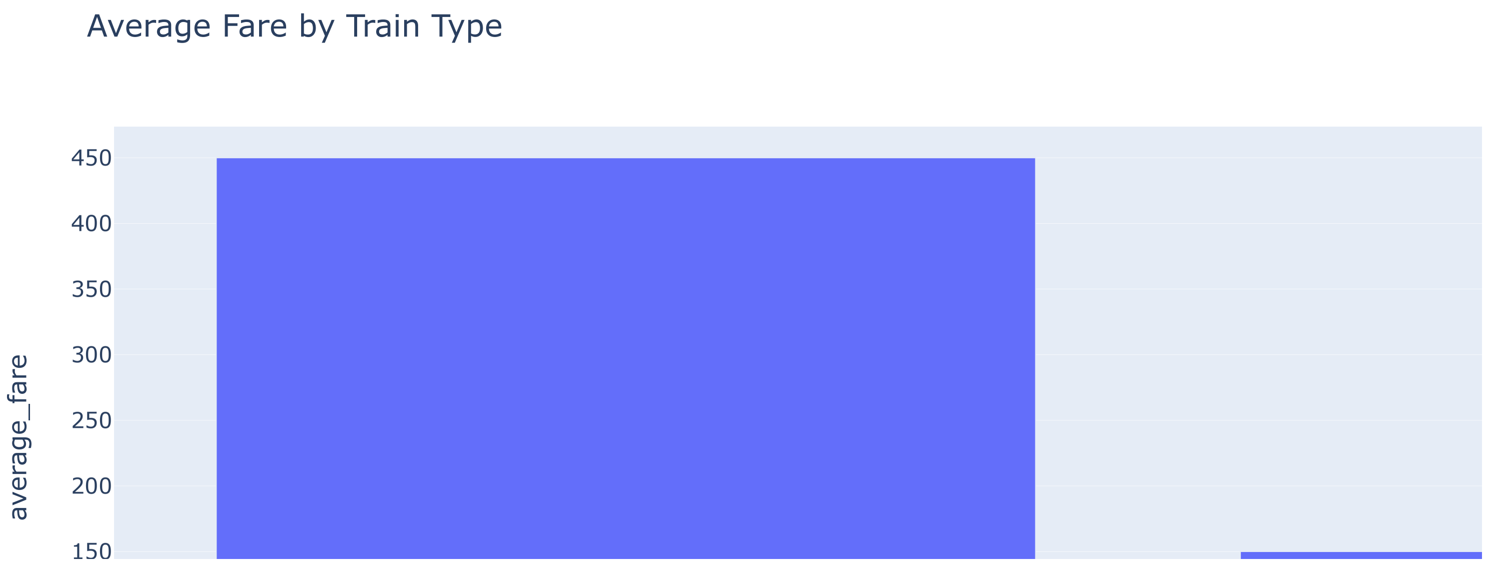
```
except Error as e:
    print("Error while connecting to MySQL", e)

finally:
    if (connection is not None) and (connection.is_connected()):
        cursor.close()
        connection.close()
        print("MySQL connection is closed")
```

```
Connected to MySQL Server version  8.0.31
Your connected to database:  ('railwayreservationsystem',)
```

```
/Users/paragjain/opt/anaconda3/lib/python3.9/site-packages/pandas/io/sql.py:762: UserWarning: pandas only sup
port SQLAlchemy connectable(engine/connection) ordatabase string URI or sqlite3 DBAPI2 connectionother DBAPI2
objects are not tested, please consider using SQLAlchemy
  warnings.warn(
/Users/paragjain/opt/anaconda3/lib/python3.9/site-packages/pandas/io/sql.py:762: UserWarning: pandas only sup
port SQLAlchemy connectable(engine/connection) ordatabase string URI or sqlite3 DBAPI2 connectionother DBAPI2
objects are not tested, please consider using SQLAlchemy
  warnings.warn(
```

Number of Tickets by Train Type





```
MySQL connection is closed
```

```
In [ ]:
```

```
In [ ]:
```

In [1]:

```
import pymongo

# Create a MongoDB client
client = pymongo.MongoClient("mongodb://localhost:27017/")

# Create the database
db = client["railway_reservation_system"]

# Create the collections
customers = db["customers"]
trains = db["trains"]
routes = db["routes"]
employees = db["employees"]
management = db["management"]
stations = db["stations"]
tickets = db["tickets"]
fares = db["fares"]
schedules = db["schedules"]

# Define the schema for the collections
customer_schema = {
    "customer_id": int,
    "name": str,
    "email": str,
    "phone_number": str,
    "address": str
}

train_schema = {
    "trainid": int,
    "type": str,
    "name": str,
    "seats": int
}

route_schema = {
    "route_id": int,
    "trainid": int,
    "from_station": str,
    "to_station": str,
    "distance": int
}
```

```
employee_schema = {
    "employee_id": int,
    "name": str,
    "address": str,
    "phone": str,
    "salary": float
}

management_schema = {
    "management_id": int,
    "employee_id": int,
    "name": str,
    "email": str,
    "phone_number": str,
    "address": str
}

station_schema = {
    "station_id": int,
    "route_id": int,
    "name": str,
    "location": str,
    "code": str
}

ticket_schema = {
    "ticket_id": int,
    "trainid": int,
    "class": str,
    "departure_date": str,
    "customer_id": int
}

fare_schema = {
    "fare_id": int,
    "ticket_id": int,
    "class": str
}

schedule_schema = {
    "schedule_id": int,
    "management_id": int,
    "trainid": int,
    "departure_time": str,
    "arrival_time": str
```

```
}

# Set up the references between the collections
routes.create_index([("trainid", pymongo.ASCENDING)])
stations.create_index([("route_id", pymongo.ASCENDING)])
tickets.create_index([("trainid", pymongo.ASCENDING)])
tickets.create_index([("customer_id", pymongo.ASCENDING)])
fares.create_index([("ticket_id", pymongo.ASCENDING)])
schedules.create_index([("management_id", pymongo.ASCENDING)])
schedules.create_index([("trainid", pymongo.ASCENDING)])

# Insert example data into the collections
customers.insert_one({"customer_id": 1, "name": "John Doe", "email": "john@example.com", "phone_number": "123-
trains.insert_one({"trainid": 1, "type": "Express", "name": "Express 1", "seats": 100})
routes.insert_one({"route_id": 1, "trainid": 1, "from_station": "Station A", "to_station": "Station B", "dista
employees.insert_one({"employee_id": 1, "name": "Jane Smith", "address": "456 Main St", "phone": "234-567-890
management
```

Out[1]:

```
Collection(Database(MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True),
'railway_reservation_system'), 'management')
```

In [2]:

```
customers.insert_one({"customer_id": 2, "name": "Parag Jain", "email": "parag@example.com", "phone_number": "
```

Out[2]:

```
<pymongo.results.InsertOneResult at 0x7f840a13ce00>
```

In [5]:

```
from pymongo import MongoClient
import random
import string


# generate dummy data for each schema
for i in range(20):
    # generate dummy data for customers
    customer_data = {
        "customer_id": i + 1,
        "name": ''.join(random.choices(string.ascii_uppercase, k=5)),
        "email": ''.join(random.choices(string.ascii_lowercase, k=5)) + "@example.com",
        "phone_number": ''.join(random.choices(string.digits, k=10)),
        "address": ''.join(random.choices(string.ascii_uppercase + string.digits, k=10))
    }
    customers.insert_one(customer_data)

    # generate dummy data for trains
    train_data = {
        "trainid": i + 1,
        "type": ''.join(random.choices(["local", "express"])),
        "name": ''.join(random.choices(string.ascii_uppercase, k=5)),
        "seats": random.randint(50, 100)
    }
    trains.insert_one(train_data)

    # generate dummy data for routes
    route_data = {
        "route_id": i + 1,
        "trainid": random.randint(1, 20),
        "from_station": ''.join(random.choices(string.ascii_uppercase, k=5)),
        "to_station": ''.join(random.choices(string.ascii_uppercase, k=5)),
        "distance": random.randint(100, 500)
    }
    routes.insert_one(route_data)
```

In [6]:

```
employee_data = [
    {"employee_id": 1, "name": "John Doe", "address": "123 Main St", "phone": "555-1234", "salary": 50000.0},
    {"employee_id": 2, "name": "Jane Smith", "address": "456 Oak Ave", "phone": "555-5678", "salary": 60000.0
    {"employee_id": 3, "name": "Bob Johnson", "address": "789 Elm St", "phone": "555-9012", "salary": 55000.0
    {"employee_id": 4, "name": "Alice Lee", "address": "321 Maple St", "phone": "555-3456", "salary": 65000.0
    {"employee_id": 5, "name": "David Kim", "address": "654 Pine St", "phone": "555-7890", "salary": 70000.0}
]
db.employees.insert_many(employee_data)

# management
management_data = [
    {"management_id": 1, "employee_id": 1, "name": "John Doe", "email": "johndoe@example.com", "phone_number"
    {"management_id": 2, "employee_id": 2, "name": "Jane Smith", "email": "janesmith@example.com", "phone_num
    {"management_id": 3, "employee_id": 3, "name": "Bob Johnson", "email": "bobjohnson@example.com", "phone_nu
    {"management_id": 4, "employee_id": 4, "name": "Alice Lee", "email": "alicelee@example.com", "phone_numbe
    {"management_id": 5, "employee_id": 5, "name": "David Kim", "email": "davidkim@example.com", "phone_numbe
]
db.management.insert_many(management_data)

# station
station_data = [
    {"station_id": 1, "route_id": 1, "name": "Station A", "location": "City X", "code": "AXY"},
    {"station_id": 2, "route_id": 1, "name": "Station B", "location": "City Y", "code": "BYX"},
    {"station_id": 3, "route_id": 2, "name": "Station C", "location": "City Z", "code": "CZY"},
    {"station_id": 4, "route_id": 2, "name": "Station D", "location": "City X", "code": "DXZ"},
    {"station_id": 5, "route_id": 3, "name": "Station E", "location": "City Y", "code": "EYX"}
]
db.stations.insert_many(station_data)
```

Out[6]:

```
<pymongo.results.InsertManyResult at 0x7f83e87bcec0>
```

In [10]:

```
from datetime import datetime
for i in range(5):
    ticket = {
        "ticket_id": i+1,
        "trainid": random.randint(1, 10),
        "class": random.choice(['Economy', 'Business', 'First']),
        "departure_date": datetime.now().strftime("%Y-%m-%d"),
        "customer_id": random.randint(1, 20)
    }
    tickets.append(ticket)

# Dummy data for fare_schema
fares = []
for i in range(5):
    fare = {
        "fare_id": i+1,
        "ticket_id": i+1,
        "class": random.choice(['Economy', 'Business', 'First'])
    }
    fares.append(fare)
```

In [12]:

```
dummy_data = [
    {
        "schedule_id": 1,
        "management_id": 101,
        "trainid": 1,
        "departure_time": "2022-05-01 08:00:00",
        "arrival_time": "2022-05-01 16:00:00"
    },
    {
        "schedule_id": 2,
        "management_id": 102,
        "trainid": 2,
        "departure_time": "2022-05-02 10:00:00",
        "arrival_time": "2022-05-02 20:00:00"
    },
    {
        "schedule_id": 3,
        "management_id": 103,
            "trainid": 3,
            "departure_time": "2022-05-03 12:00:00",
            "arrival_time": "2022-05-03 22:00:00"
        },
        {
            "schedule_id": 4,
            "management_id": 104,
            "trainid": 4,
            "departure_time": "2022-05-04 14:00:00",
            "arrival_time": "2022-05-04 23:00:00"
        },
        {
            "schedule_id": 5,
            "management_id": 105,
            "trainid": 5,
            "departure_time": "2022-05-05 16:00:00",
            "arrival_time": "2022-05-05 01:00:00"
        }
]

db.schedules.insert_many(dummy_data)
```

Out[12]:

```
<pymongo.results.InsertManyResult at 0x7f83e87c0d80>
```

In [13]:

```
# Adding dummy data to the ticket schema
ticket_data = [
    {"ticket_id": 1, "trainid": 1234, "class": "Economy", "departure_date": "2023-05-15", "customer_id": 101}
    {"ticket_id": 2, "trainid": 5678, "class": "First Class", "departure_date": "2023-06-23", "customer_id":
    {"ticket_id": 3, "trainid": 9012, "class": "Business", "departure_date": "2023-07-30", "customer_id": 103
    {"ticket_id": 4, "trainid": 3456, "class": "Economy", "departure_date": "2023-08-18", "customer_id": 104}
    {"ticket_id": 5, "trainid": 7890, "class": "First Class", "departure_date": "2023-09-25", "customer_id":
]

db.tickets.insert_many(ticket_data)

# Adding dummy data to the fare schema
fare_data = [
    {"fare_id": 1, "ticket_id": 1, "class": "Economy"},
    {"fare_id": 2, "ticket_id": 2, "class": "First Class"},
    {"fare_id": 3, "ticket_id": 3, "class": "Business"},
    {"fare_id": 4, "ticket_id": 4, "class": "Economy"},
    {"fare_id": 5, "ticket_id": 5, "class": "First Class"}
]

db.fares.insert_many(fare_data)
```

Out[13]:

```
<pymongo.results.InsertManyResult at 0x7f83e8391e40>
```

In [18]:

```python
#Query 1 to get all customers who have booked tickets for a particular train:
train_id = 3
tickets = db.tickets.find({'trainid': train_id})
customer_ids = [ticket['customer_id'] for ticket in tickets]
customers = db.customers.find({'customer_id': {'$in': customer_ids}})
print(f'Query to get all customers who have booked tickets for a particular: {customers}')

#This query retrieves all the tickets booked for a specific train and extracts the customer IDs from the ticke
```

```
Query to get all customers who have booked tickets for a particular: <pymongo.cursor.Cursor object at 0x7f840
a13a130>
```

In [17]:

```python
#Query 2 to get the total distance covered by a particular train:
train_id = 7
routes = db.routes.find({'trainid': train_id})
total_distance = sum([route['distance'] for route in routes])
print(f'Total distance covered by train {train_id}: {total_distance} km')

#This query retrieves all the routes covered by a specific train and calculates the total distance covered by
```

```
Total distance covered by train 7: 0 km
```

In [19]:

```python
#Query 3 to get the average salary of employees:
avg_salary = db.employees.aggregate([
    {
        '$group': {
            '_id': None,
            'avg_salary': {'$avg': '$salary'}
        }
    }
]).next()['avg_salary']

print(f'Average employee salary: {avg_salary}')


#This query uses the $group aggregation operator to group all employees together and calculate the average sa
```

```
Average employee salary: 57142.857142857145
```

In [21]:

```python
#Query 4 to get the name and location of all stations on a particular route:
route_id = 3
stations = db.stations.find({'route_id': route_id}, {'_id': 0, 'name': 1, 'location': 1})
for station in stations:
    print(station)
#This query finds all the stations on a particular route and returns their names and locations.
```

```
{'name': 'Station E', 'location': 'City Y'}
```

In [26]:

```
#Query 5 to get the number of tickets sold for each class on a particular train:
train_id = 5
tickets = db.tickets.find({'trainid': train_id})
class_counts = {}
for ticket in tickets:
    class_ = ticket['class']
    if class_ not in class_counts:
        class_counts[class_] = 0
    class_counts[class_] += 1
print(f'Ticket sales for train {train_id}:')
for class_, count in class_counts.items():
    print(f'{class_}: {count}')


#This query finds the total number of tickets sold for each class on particular train. \
#This helps determine each class's tickets sold
```

```
Ticket sales for train 5:
First Class
```

In [ ]: