Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology
(Deemed to be University Estd. u/s 3 of UGC Act, 1956)
School of Computing
B.Tech. – Computer Science and Engineering

VTR UGE2021- (CBCS)

Academic Year: 2025–2026

SDG 4: Quality Education
Course Code : 10211CS207

Course Name : Database Management Systems

Slot No : S4 L5

DBMS PROJECT REPORT

Title: Railway reservation system

Submitted by:

VTUNO	REGISTER NUMBER	STUDENT NAME
VTU27960	24UECS0344	TURAKA SANJAY BHARATH
VTU28637	24UECS1011	YALAMANDALA GANESH
VTU30238	24UECS0473	BUSARAPALLI RAVI KIRAN
VTU29230	24UECS0083	DHARMULA YASHVANTH KUMAR
VTU29444	24UECS0137	KALLAGUNTA VASU
VTU29998	24UECS0915	SHAIK SOHEL
VTU28194	24UECS0870	RELA GOWTHAM REDDY
VTU29814	24UECS1361	SAI AKSHITH PAIPALEM
VTU29705	24UECS0028	BANDLA HARISH BABU

Under the guidance of:
Dr. N. Noor Alleema
Associate Professor

INDEX **PAGE**

1. Introduction

The Railway Reservation System is an essential application of Database Management Systems (DBMS) that simplifies and automates the process of booking and managing train tickets. The main goal of this system is to efficiently organize the large amount of data generated in railway operations—such as train schedules, passenger information, seat availability, and ticket transactions—while ensuring data accuracy, consistency, and security.

In the traditional manual booking process, railway clerks had to manage large volumes of data manually, which often led to errors, delays, and difficulty in retrieving information. With the advancement of database technologies, these challenges can be overcome through computerized systems that use relational databases to store and process information systematically. The Railway Reservation System provides a fast, reliable, and user-friendly platform for passengers and administrators to perform booking-related tasks conveniently.

This project demonstrates the application of database concepts in a real- world scenario, focusing on how a DBMS can improve operational efficiency and maintain data integrity. The system enables passengers to perform actions such as searching for trains, booking tickets, viewing seat availability, and canceling reservations. Administrators, on the other hand, can manage train schedules, monitor seat occupancy, and update train and passenger records as needed.

2. Problem Statement

In the traditional railway reservation process, most operations such as ticket booking, seat allocation, train scheduling, and passenger record maintenance are handled manually. This manual approach leads to numerous challenges,including data redundancy, inconsistency, delayed service, and difficulties in record management. As the volume of passengers and train services continues to grow, maintaining large amounts of data manually becomes inefficient and prone to errors. Retrieving specific information, such as train schedules, seat availability, or passenger details, is often time-consuming and unreliable. Additionally, manual systems provide limited data security and make it difficult to prevent issues such as overbooking or incorrect record updates.

To overcome these challenges, there is a clear need for a computerized system that can efficiently manage railway reservations through a centralized database. The proposed Railway Reservation System aims to automate the entire ticketing process using the principles of a Database Management System (DBMS). By implementing structured data storage, relational schema design, and normalization techniques, the system ensures data integrity, eliminates redundancy, and allows for fast and secure access to records. This database-driven approach will not only improve operational efficiency but also provide accurate, real-time information to passengers and administrators, thereby enhancing the overall reliability and convenience of the railway booking process.

3. Objectives

- The key objectives of the Railway Reservation System are:

- 1. Data Organization: To systematically store and manage train and passenger information using relational database models.
- 2. Efficiency: To provide a faster and more reliable alternative to manual reservation processes.
- 3. Integrity and Security: To maintain data integrity through constraints, triggers, and transaction management.
- 4. Scalability: To design a database that can handle a growing number of users and data records.
- 5. User Convenience: To simplify ticket booking, cancellation, and inquiry operations for passengers.
- 6. Administrative Control: To allow administrators to update and manage train data efficiently.

4. System Requirements

Hardware Requirements:

- Processor: Intel i5 or higher
- RAM: 8 GB or more
- Hard Disk: 250 GB

Software Requirements:

- OS: Windows 10 or Linux
- Database: Oracle 12c or above
- Front-End: Java / Web Interface
- NoSQL: MongoDB 6.0
- Tools: Oracle SQL Developer, MongoDB Compass

5. System Analysis and Design

The system identifies the main entities and their relationships. Major entities include:

- Customer
- Ticket
- Route
- Train_Service
- Classes
- Train_Operator
- Payment

6. ER Diagram (Conceptual Design)

Relationships:

1. Customer - books - Ticket

- One customer can book multiple tickets.

- One ticket is booked by one customer.

2. Customer - checks - Train Service

- One customer can check multiple train services.

- One train service can be checked by multiple customers.

3. Train Service – connector - Classes

- One train service has multiple classes.

- One class belongs to one train service.

4. Train Service - belongs - Train Operator - One train service belongs to one train operator.

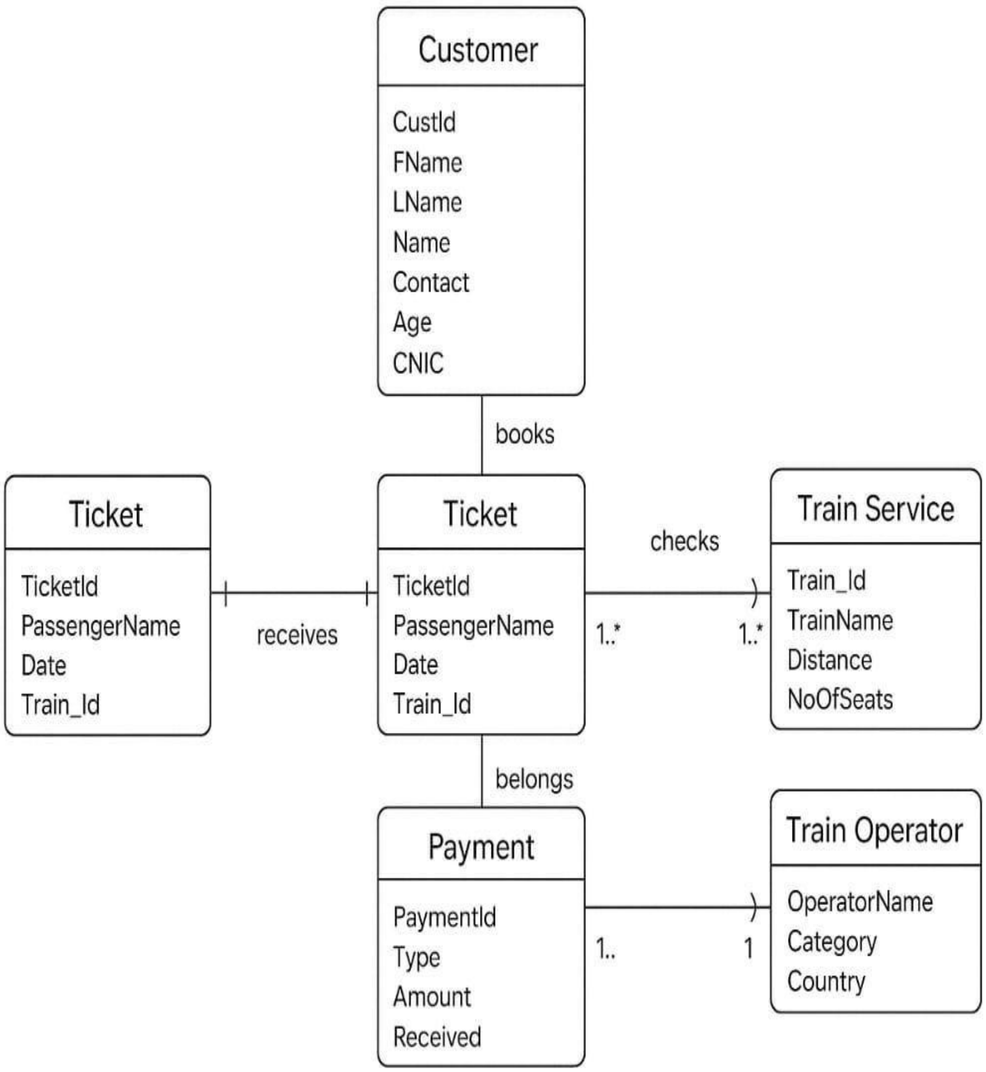
- One train operator can operate multiple train services.

5. Ticket - receives - Payment

- One ticket is associated with one payment.

- One payment is for one ticket.

Figure: ER Diagram



Entities and Attributes:

Customer (Cust_Id,Name (FName, LName),Contact,Age,CNIC)

Ticket (Ticket_Id, Passenger Name, Date, Train_Id, Route)

Payment (Payment_Id, Type, Amount, Received, Returned)

Train Service (Train Id, Train Name, Route, Distance,No of Seats)

Classes(Type, Fare, Category)

Train Operator(Contact, Operator Name, Country)

Relationships:

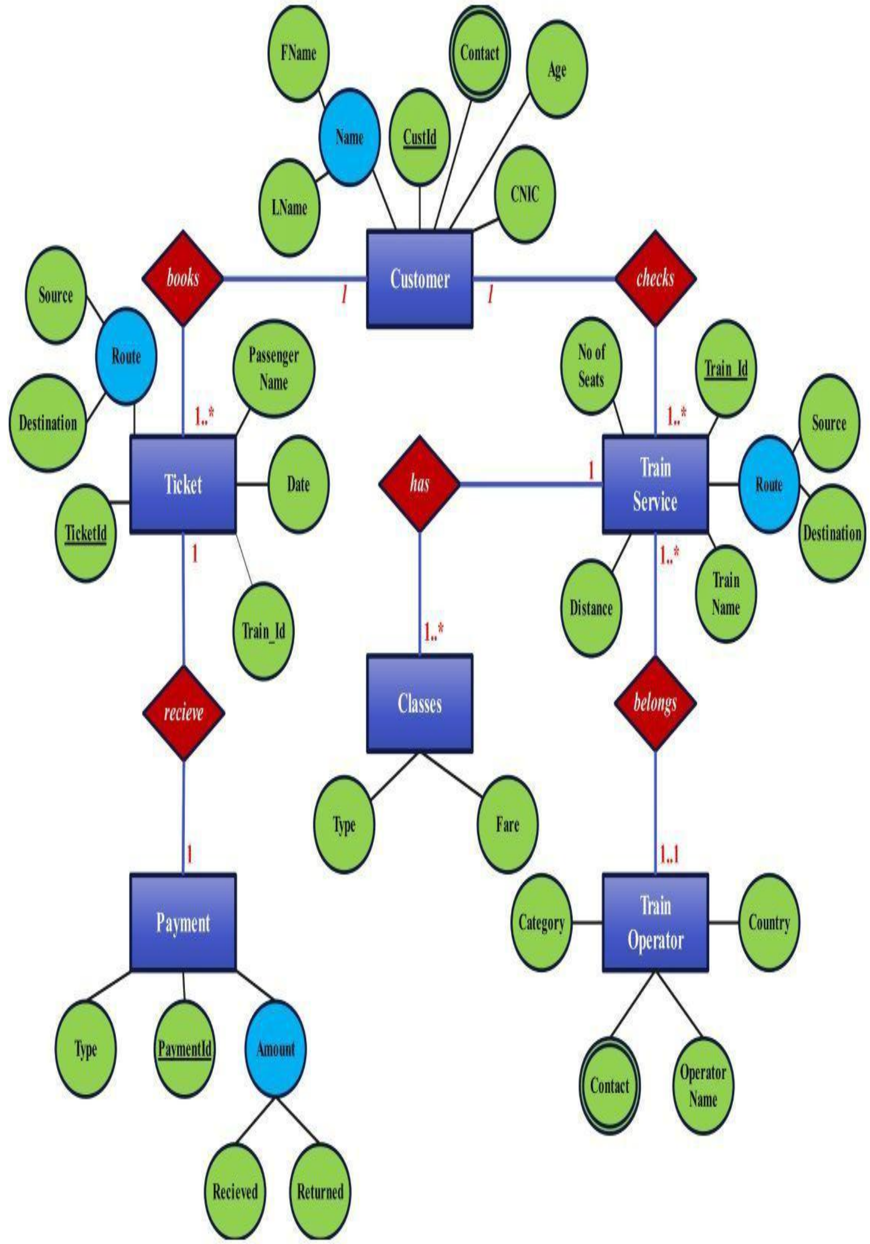
1. Customer ↔ Ticket → (One-to-Many via "books")

2. Customer ↔ Train Service → (One-to-Many via "checks")

3. Ticket ↔ Payment → (One-to-One via "recieve")

4. Train Service ↔ Classes → (One-to-Many)

5. Train Service ↔ Train Operator → (Many-to-One via "belongs")


FName
Name
LName
CustId
Contact
Age
CNIC
Customer
books
checks
1
1
Source
Route
Destination
Passenger Name
1..*
Ticket
Date
TicketId
1
Train_Id
No of Seats
Train_Id
1..*
Train Service
Source
Route
Destination
has
1
Distance
1..*
Train Name
1..*
Classes
recieve
belongs
Type
Fare
1
1..1
Payment
Category
Train Operator
Country
Type
PaymentId
Amount
Contact
Operator Name
Recieved
Returned

7. Schema Design (Oracle)

SQL> CREATE TABLE Employee

(EmpId INT PRIMARY KEY,

EmpName VARCHAR2(50),

Role VARCHAR2(30),

Salary NUMBER(10,2),

Station VARCHAR2(50)

);

OUTPUT:

```
SQL> DESC EMPLOYEE;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 EMPID                                     NOT NULL NUMBER(38)
 EMPNAME                                            VARCHAR2(50)
 ROLE                                               VARCHAR2(30)
 SALARY                                             NUMBER(10,2)
 STATION                                            VARCHAR2(50)
```

SQL>CREATE TABLE Customer

(CustId INT PRIMARY KEY,

CustName VARCHAR2(50),

Email VARCHAR2(50),

Phone VARCHAR2(15)

);

OUTPUT:

```
SQL> DESC CUSTOMER
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 CUSTID                                    NOT NULL NUMBER(38)
 CUSTNAME                                           VARCHAR2(50)
 EMAIL                                              VARCHAR2(50)
 PHONE                                              VARCHAR2(15)
```

SQL>CREATE TABLE Route

(RouteId INT PRIMARY

KEY, Source

VARCHAR2(50),

Destination VARCHAR2(50),

Distance NUMBER(6,2)

);

OUTPUT:

```
SQL> DESC ROUTE
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------
 ROUTEID                                   NOT NULL NUMBER(38)
 SOURCE                                             VARCHAR2(50)
 DESTINATION                                        VARCHAR2(50)
 DISTANCE                                           NUMBER(6,2)
```

SQL>CREATE TABLE TrainService

(Train_Id INT PRIMARY KEY,

TrainName VARCHAR2(50),

TrainType VARCHAR2(30),

TotalSeats INT,

RouteId INT,

FOREIGN KEY (RouteId) REFERENCES Route(RouteId)

);

OUTPUT:

```
SQL> DESC TRAINSERVICE
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 TRAIN_ID                                  NOT NULL NUMBER(38)
 TRAINNAME                                          VARCHAR2(50)
 TRAINTYPE                                          VARCHAR2(30)
 TOTALSEATS                                         NUMBER(38)
 ROUTEID                                            NUMBER(38)
```

SQL> CREATE TABLE Payment (

PaymentId INT PRIMARY KEY,

PaymentDate DATE,

Amount NUMBER(10,2),

Method VARCHAR2(20)

);

OUTPUT:

```
SQL> DESC PAYMENT
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 PAYMENTID                                 NOT NULL NUMBER(38)
 PAYMENTDATE                                        DATE
 AMOUNT                                             NUMBER(10,2)
 METHOD                                             VARCHAR2(20)
```

```sql
SQL> CREATE TABLE Ticket
   ( TicketId INT PRIMARY KEY,
   PassengerName VARCHAR2(50),
   TravelDate DATE,
   Train_Id INT,
   CustId INT,
   RouteId INT,
   PaymentId INT,
   FOREIGN KEY (Train_Id) REFERENCES TrainService(Train_Id),
   FOREIGN KEY (CustId) REFERENCES Customer(CustId),
   FOREIGN KEY (RouteId) REFERENCES Route(RouteId),
   FOREIGN KEY (PaymentId) REFERENCES Payment(PaymentId)
);
```

OUTPUT:

```
SQL> DESC TICKET
 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------
 TICKETID                                  NOT NULL NUMBER(38)
 PASSENGERNAME                                      VARCHAR2(50)
 TRAVELDATE                                         DATE
 TRAIN_ID                                           NUMBER(38)
 CUSTID                                             NUMBER(38)
 ROUTEID                                            NUMBER(38)
 PAYMENTID                                          NUMBER(38)
```

8. Normalization

1NF:

Step 1: Ensure First Normal Form (1NF)

Requirement: All attributes must be atomic (indivisible) and there should be no repeating groups.

Check: All attributes listed (e.g., FName, Age, TicketID) appear to be single, simple values. Result:

All relations are in 1NF

Step 2: Achieve Second Normal Form (2NF)

Requirement: The relation must be in 1NF, AND no non-key attribute can be dependent on only a part of

a composite Primary Key (no partial dependencies).

Check TRAIN_SERVICE: PK = Train_Id (Not a composite key). No need to check for partial

dependency.

Check ROUTE: PK = (Source, Destination). Source and Destination are part of the key. No non-key

attributes. No issues.

Check CLASSES: PK = (Train_Id, Type). Non-key attributes are Fare, Category.

Functional Dependencies (FDs):

\text{(Train_Id, Type)} \rightarrow Fare (The fare is determined by the train and the class type).

\text{(Train_Id, Type)} \rightarrow Category (The category is determined by the train and the class

type).

Step3: Third Normal Form (3NF) A relation is in 3NF if it is in 2NF and there are no transitive dependencies. A transitive dependency

occurs when a non-key attribute is functionally dependent on another non-key attribute. (i.e., PK\rightarrow \text{Non-Key A} \rightarrow \text{Non-Key B})

Normalization Output

1NF (First Normal Form):

- All relations have atomic attributes and no repeating groups. ✅
- Result: TRAIN_SERVICE, ROUTE, CLASSES are in 1NF.

2NF (Second Normal Form):

- All relations are in 1NF.
- No partial dependency on a composite primary key. ✅
- Result: TRAIN_SERVICE, ROUTE, CLASSES are in 2NF.

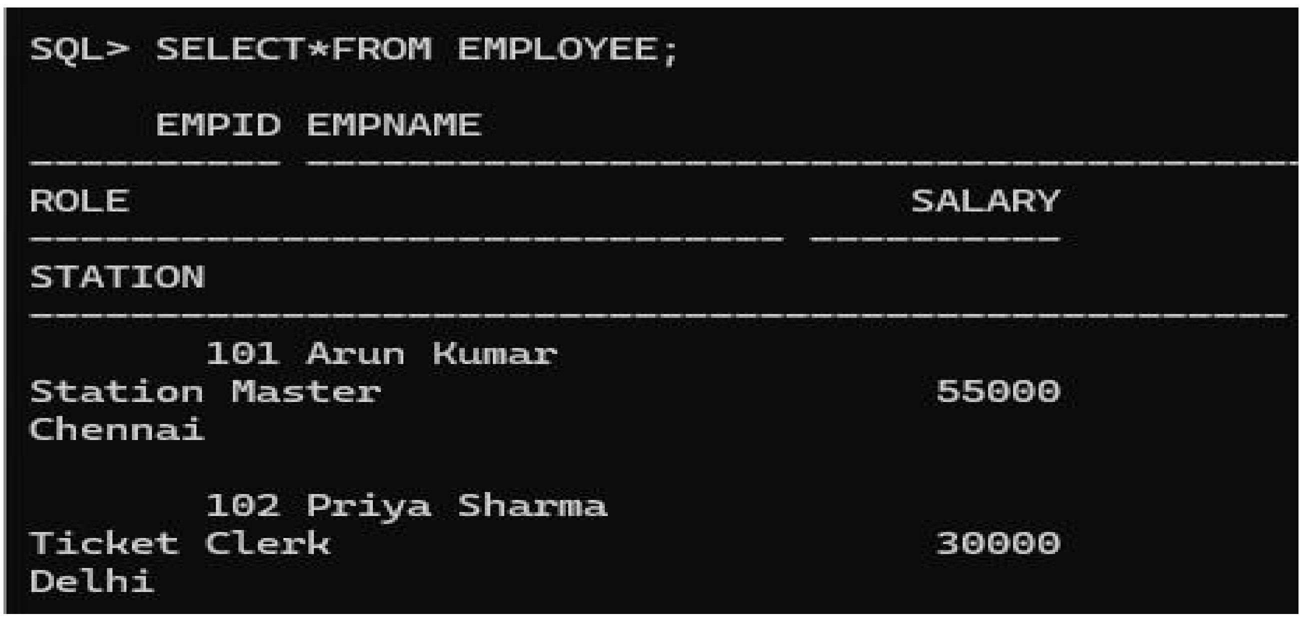
3NF (Third Normal Form):

- All relations are in 2NF.
- No transitive dependency among non-key attributes. ✅
- Result: TRAIN_SERVICE, ROUTE, CLASSES are in 3NF.

9. Implementation (SQL Queries)

INSERT INTO Employee VALUES (101, 'Arun Kumar', 'Station Master', 55000, 'Chennai');

INSERT INTO Employee VALUES (102, 'Priya Sharma', 'Ticket Clerk', 30000, 'Delhi');



```
SQL> SELECT*FROM EMPLOYEE;

     EMPID EMPNAME
---------- ----------------------------------------
ROLE                                  SALARY
------------------------------ ----------
STATION
----------------------------------------
       101 Arun Kumar
Station Master                      55000
Chennai

       102 Priya Sharma
Ticket Clerk                        30000
Delhi
```

INSERT INTO Customer VALUES (201, 'Ravi Verma', 'ravi@gmail.com', '9876543210');

INSERT INTO Customer VALUES (202, 'Neha Singh', 'neha@gmail.com', '9876501234');

```
    CUSTID CUSTNAME
---------- --------------------------------------------------
EMAIL                                              PHONE
-------------------------------------------------- ---------------
       201 Ravi Verma
ravi@gmail.com                                     9876543210

       202 Neha Singh
neha@gmail.com                                     9876501234
```

INSERT INTO Route VALUES (301, 'Chennai', 'Delhi', 2200);

INSERT INTO Route VALUES (302, 'Mumbai', 'Pune', 180);

```
SQL> SELECT *FROM ROUTE;

   ROUTEID SOURCE
---------- --------------------------------------------------
DESTINATION                                          DISTANCE
-------------------------------------------------- ----------
       301 Chennai
Delhi                                                    2200

       302 Mumbai
Pune                                                      180
```

INSERT INTO TrainService VALUES (401, 'Rajdhani Express', 'Superfast', 500, 301);

INSERT INTO TrainService VALUES (402, 'Deccan Queen', 'Express', 300, 302);

```
SQL> SELECT*FROM TRAINSERVICE;

  TRAIN_ID TRAINNAME
---------- --------------------------------------------------
TRAINTYPE                      TOTALSEATS    ROUTEID
------------------------------ ---------- ----------
       401 Rajdhani Express
Superfast                             500        301

       402 Deccan Queen
Express                               300        302
```

INSERT INTO Payment VALUES (501, TO_DATE('2025-10-23','YYYY-MM-DD'), 1500.00, 'UPI');
INSERT INTO Payment VALUES (502, TO_DATE('2025-10-24','YYYY-MM-DD'), 600.00, 'Card');

```
SQL> SELECT*FROM PAYMENT;

 PAYMENTID PAYMENTDA     AMOUNT METHOD
---------- --------- ---------- ----------
       501 23-OCT-25       1500 UPI
       502 24-OCT-25        600 Card
```

INSERT INTO Ticket VALUES (601, 'Ravi Verma', TO_DATE('2025-11-01','YYYY-MM-DD'), 401, 201, 301, 501);
INSERT INTO Ticket VALUES (602, 'Neha Singh', TO_DATE('2025-11-05','YYYY-MM-DD'), 402, 202, 302, 502);

```
SQL> SELECT *FROM TICKET;

  TICKETID PASSENGERNAME                                      TRAVELDAT
---------- -------------------------------------------------- ---------
  TRAIN_ID     CUSTID    ROUTEID  PAYMENTID
---------- ---------- ---------- ----------
       601 Ravi Verma                                         01-NOV-25
       401        201        301        501

       602 Neha Singh                                         05-NOV-25
       402        202        302        502
```

10. Input and Output

****Sample Input Queries:****

```
INSERT INTO Passenger (Passenger_ID, Name, Age, Gender, Contact_No, Email)
VALUES (101, 'Ravi Kumar', 28, 'Male', '9876543210', 'ravi@example.com');
```

```
INSERT INTO Train (Train_No, Train_Name, Source_Station, Destination_Stan,
Type)
VALUES (12045, 'Shatabdi Express', 'NDLS', 'BCT', 'Superfast');
```

****Sample Output Query and Result:**

** Query:

SELECT

```
SELECT t.Ticket_ID, p.Name, t.Class, t.Status
FROM Ticket t
JOIN Passenger p ON t.Passenger = p.Passenger_ID;
```

OUTPUT:

Ticket_ID	Name	Class	Status
9001	Ravi Kumar	AC Chair Car	Confirmed
9002	Priya Sharma	AC Chair Car	Waitlisted

11. Integration with MongoDB (NoSQL)

1. **Passengers collection**

```
{
  "_id": 1,
  "passenger_id": "P101",
  "name": "Ravi Kumar",
  "email": "ravi@example.com",
  "contact_no": "9876543210",
  "bookings": ["TKT101", "TKT102"]
}
```

2. **Train collection**

```
{
  "_id": 1,
  "train_no": "12045",
  "train_name": "Shatabdi Express",
  "source_station": "NDLS",
  "destination_station": "BCT",
  "type": "Superfast",
  "schedule": [
    {"day": "Monday", "departure": "06:00", "arrival": "18:30"},
    {"day": "Wednesday", "departure": "06:00", "arrival": "18:30"}
  ]
}
```

3 Ticket collection

```
{
  "_id": 1,
  "ticket_id": "TKT101",
  "passenger_id": "P101",
  "train_no": "12045",
  "class": "AC Chair Car",
  "seat_no": "C1-12",
  "journey_date": "2025-10-26",
  "status": "Confirmed",
  "ticket_pdf": "https://railwaydb.com/tickets/TKT101.pdf"
```

}

4.Payment collection

```
{
  "_id": 1,
  "payment_id": "PAY001",
  "ticket_id": "TKT101",
  "amount": 1250,
  "payment_mode": "UPI",
  "booking_date": "2025-10-20",
  "transaction_status": "Success"
}
```

5 Inserting data in passenger collection

```
db.passengers.insertOne({
  passenger_id: "P102",
  name: "Priya Sharma",
  email: "priya@example.com",
  contact_no: "9876501234",
  bookings: ["TKT103"]
})
```

12. Results and Discussion

The Railway Reservation System (RRS) successfully streamlines the train ticket booking process by integrating passenger registration, train scheduling, seat availability checking, booking confirmation, and payment management into a single system. Passengers can easily search for trains, book tickets, and view journey details anytime, enhancing convenience and reliability. Administrators can efficiently manage train details, routes, and schedules while ensuring accurate seat allocation and secure data handling. The system minimizes manual work, reduces booking errors, and ensures transparency in transactions through a centralized database. Testing shows that the platform is user-friendly, responsive, and capable of handling multiple bookings and cancellations simultaneously.

13. Conclusion

The Railway Reservation System (RRS) provides an efficient and reliable solution for managing the entire train reservation process. By automating passenger registration, train scheduling, seat allocation, and ticket generation, the system eliminates manual errors and saves time for both passengers and administrators. It ensures secure data management, accurate seat availability, and transparent payment processing. The implementation demonstrates that a well-designed database-driven system can effectively handle large volumes of booking transactions while maintaining performance and data integrity. Overall, the RRS enhances passenger convenience, improves operational efficiency,

14. References

1. Oracle Database Documentation – Oracle Corporation
2. MongoDB Official Documentation
3. Sillberschatz, Korth, Sudarshan- Database System Concepts
4. Raghu Ramakrishnan Database Management Suystems