Vel Tech
Rangarajan Dr. Sagunthala
R&D Institute of Science and Technology

School of Computing
Department of Computer Science & Engineering
(Artificial Intelligence and Machine Learning)

ACADEMIC YEAR 2025-26 (SUMMER SEMESTER)

LAB RECORD NOTEBOOK

10211CA207 – DATABASE MANAGEMENT SYSTEMS

NAME: K. Soomanath Reddi

VTU.NO: 27832

REG.NO: 24UECL0027

BRANCH: CSE (AI & ML)

YEAR/SEM: 2nd year / 3rd Sem

SLOT: S10 L12

INDEX

Vel Tech
Rangarajan Dr. Sagunthala
R&D Institute of Science and Technology

School of Computing
Department of Computer Science & Engineering
(Artificial Intelligence and Machine Learning)

ACADEMIC YEAR 2025-26 (SUMMER SEMESTER)

BONAFIDE CERTIFICATE

NAME :	K. Soomanath Reddi	**BRANCH** :	CSE (AI & ML)
VTU NO. :	27832	**REG.NO.** :	24UECL0027
YEAR/SEM :	2nd year / 3rd Sem	**SLOT NO.** :	S10 L12

Certified that this is a bonafide record of work done by above student in the **"10211CA207-DATABASE MANAGEMENT SYSTEMS LABORATORY"** during the year 2025-2026 (Summer Semester).

R.T. Thyagarajan 30/10/25

SIGNATURE OF LAB HANDLING FACULTY

31/10/25

SIGNATURE OF HOD

Submitted for the Semester Practical Examination held on **04.11.25** at Vel Tech Rangarajan Dr.Sagunthala R&D Institute of Science and Technology.

INTERNAL EXAMINER

EXTERNAL EXAMINER

Online Banking Management System

INDEX

S. No	Date	Title	Page No.	Marks	Faculty Signature
1.	24/07/25	Conceptual Design after Formal Technical Review	1	18	24/7
2.	31/07/25	Generating design of other traditional database models	4	18	31/7
3.	07/08/25	Developing queries with DML Single-row functions and operations	11	18	7/8
4.	14/08/25	Developing queries with DML Multi-row functions and operations	15	18	14/8
5.	21/08/25	Writing Join Queries, equivalent, and/or recursive queries	19	18	21/8
6.	28/08/25	Writing PL/SQL using Procedures, Function	22	18	28/8
7.	04/09/25	Writing PL/SQL using Loops	25	20	4/9
8.	11/09/25	Normalizing databases using functional dependencies up to BCNF	29	19	11/9
9.	18/09/25	Backing up and recovery in databases	33	18	18/9
10.	25/09/25	CRUD operations in Document databases	36	20	25/9
11.	09/10/25	CRUD operations in Graph databases	40	19	9/10
12.	16/10/25	Micro Project: Army supply chain, Bill of material and maintaince cost management completed	44	18	16/10

Total Marks: 222/240

R.S. Thakur

Signature of Faculty

Task 1: Conceptual Design after FTR (Full text Review)

using basic database design methodology and ER modeler, design Entity Relationship Diagram by satisfying the following sub tasks:

1. a Identifying the entities
1. b Identifying the attributes
1. c Identifying of relationships, cardinality, type of relationship
1. d Reframing the relations with keys and constraints.

24/7/25

Task-1 conceptual design after Full text Review

A Temple ticket online booking management system

A temple ticket online booking management system allows devotes to book tickets for temple visits, special doorshan, Poojas, and other events online, reducing physical queoues and improving crowd management. Then systems offers include features like date and time slot selection, Payment processing and the generation of tickets or Passes.

Entity :- 1. a identifying the Entities

The real-world object or concept that can be distinctly identical. Examples included student, courses, or Products.

Set :-

A collection of Entities of the same type. for instance, all students in a university would form an Entity set.

Attributes :-

A property or characteristic of an Entity. for Example, a student might have attributes like name, ID and major.

Relationship :- i.e. Identification of Relationship

An Association or interaction between two or more Entites. for Example, a student might Enroll in a course Establishing a relationship between the "student" and "course" Entities.

Task 1 :- Conceptual design after full ~~ticket~~ Review

Aim :-

To design and develop an Entity-relationship (ER) model for a temple ticket online Booking management system that allows devotes to book tickets for temple visits, special darshan, Poojas, and other events online, reducing Physical queues and improving crowd management.

Attributes :- 1.6. Identifying the attributes

* Devotes :- Devotes-ID, Name, Phone, Email, address, Age, Gender.
* Booking :- Booking-ID, Date, Time-slot, No.of. Persons, status
* Event :- Event-ID, Event-Name, Description, Price, Event-Type, duration.
* Temple :- Temple-ID, Temple-Name, location, opening hours.
* Payment :- Payment-ID, Amount, Method, status, Date
* Ticket :- Ticket-ID, Issue-Date, QR-code, validity.
* admin :- Admin-ID, Name, username, password, Role.

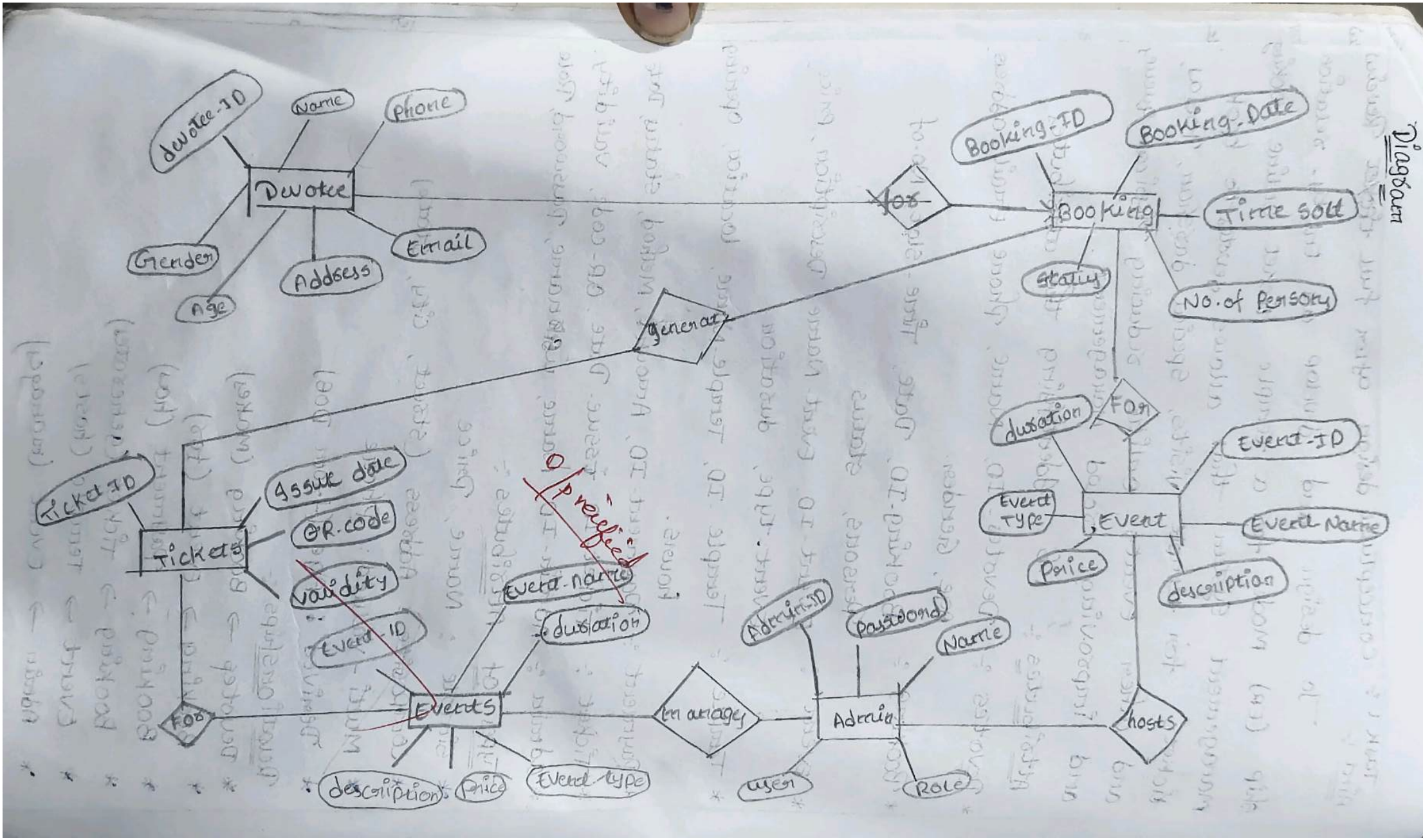
Types of Attributes :-

* Simple : Name, Price
* Composite : Address (street, city, state)
* Multi-Valued : Phone
* Derived : Age (from DOB)

Relationships :-

* Devotes → Booking (Makes)
* Booking → Event (for)
* Booking → Payment (has)
* Booking → Ticket (generates)
* Event → Temple (hosts)
* Admin → Event (manages)

Diagram


Dwotee-ID
Name
Phone
Dwotee
Gender
Age
Address
Email
for
Booking-ID
Booking-Date
Booking
Time Slot
Status
No. of Persons
generate
Ticket-ID
Issue date
QR-code
Tickets
validity
Event-ID
for
Events
description
Price
Event-type
Event-name
duration
Verified
manages
Admin-ID
password
Name
Admin
User
Role
hosts
duration
for
Event-ID
Event Type
Event
Event Name
Price
description

Relationship Types

* One-to-One : Booking → Payment, Booking → Ticket
* One-to-many : Devote → Booking
* Many-to-One : Booking → Event
* One-to-many : Temple → Event
* One to Many : Admin → Event.

cardinality :-

* 1:N and 1:1 where applicable.

VEL TECH - CSE	
EX NO	1
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	+5
TOTAL (20)	13+5=18
SIGN WITH DATE	

Thu 24/7/25

Result :-

The ER diagram for the temple ticket Booking System was successfully designed, showing all Entities, attributes, and relationship with correct cardinalities for database implementations.

Task 2: Generating design of other traditional database model.

Aim:- Creating Hiesarchical / Network model of the database by enhancing the sound abstract data by performing following tasks using forms of inheritance:

2a. Identify the specificity of each relationship, find and form surplus relations.

2b. check is.a hierarchy/ has-a hiesarchy and Performs generalization and/or specialization relationship.

2c. Find the domain of the attribute and Perform and a check constraint to the applicable

2d. Rename the relations.

2e. Perform SQL Relations using DDL, DCL commands.

2a. Identify the specificity of each relationship find and form surplus relations.

Relationship :: Temple manages Event (one-to-many).

Specificity :- One Temple can organize many events, but each Event is managed by only one temple.

Surplus Relation:- No surplus relation Needed since it is already one-to-many.

Relationship 2 :- Event has Ticket (One to many)

Specificity :- Each Event can have many ticket types (e.g, General, VIP), but each ticket type belongs to one Event.

Surplus Relation :- No surplus relation needed; one-to-many is appropriate.

Relationship 3 :- customer books ticket (many-to-many)

Specificity :- A customer can book many tickets, and a ticket can be booked by many customers (for group bookings).

Surplus Relation :- Yes, surplus relation needed for many-to-many relationship, e.g, a Booking table.

Relationship 4 :- Tickets allocated to seat (one-to-one)

Specificity :- Tickets corresponds to a specific seat, and each seat can be assigned to only one ticket.

Surplus Relation :- No surplus relation needed, one-to-one relationship is sufficient.

26. Check is-a hierarchy/has-a hierarchy and Perform generalization and/or Specialization.

Generalization. Example :-

→ Entities : Customer, Admin

→ common attributes : User_ID, First_Name, Last_Name, Email, contact_Number, password.

→ Generalized superclass: User with above attributes

→ subclasses:

Customer: Additional Attributes like customer_ID, Address, Payment_INFO

Admin: Additional Attributes like Admin_ID, Role.

Specialization Example:

→ Entity: Ticket

→ Specialized into:

General Ticket (attributes: Price, general_access_area)

Vip Ticket (attributes: Price, access_to_special_area, complimentary_services)

2. Find the domain of attributes and Perform check constraints.

Attribute	Domain	check constraint Example
age (customer)	Positive integer (minimum 18)	CHECK (age >=18)
ticket_price	Positive decimal number	CHECK (ticket_price >0)
booking_date	Date (not in Past)	CHECK (booking_date >= current_date)
seat_number	String or number within valid range	CHECK (seat_number Between 1 and 500)

2.d Rename the relations (tables) Example renaming columns for clarity:

SQL

ALTER TABLE customer RENAME COLUMN contact -no TO phone_no;

ALTER TABLE Ticket RENAME column price to ticket_price;

ALTER TABLE Booking RENAME column book_date TO booking_date;

DCL (Data control language);

SQL

-- Creating a new user and granting Privileges.

Create USER temple_user IDENTIFIED By Password 123;

GRANT CONTACT, RESOURCE To temple_user;

--- Grant Privileges on tables to temple_user;

GRANT SELECT, INSERT, UPDATE, DELETE ON customer TO temple_user;

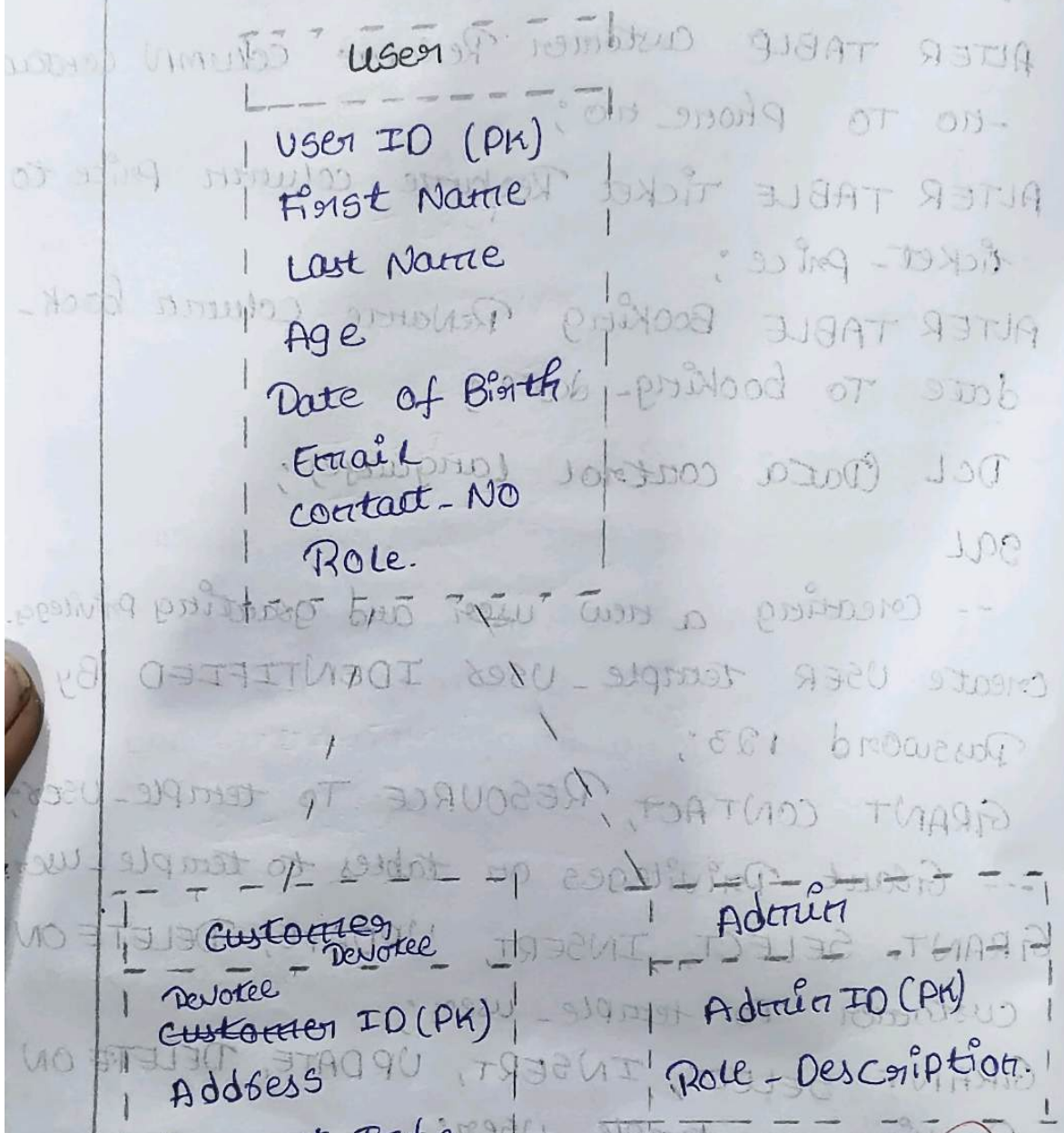
GRANT SELECT, INSERT, UPDATE, DELETE ON Booking TO temple_user;

GRANT SELECT, INSERT, UPDATE, DELETE ON ticket TO temple_user;

GRANT SELECT, INSERT, UPDATE, DELETE ON Event TO temple_user;

GRANT SELECT, INSERT, UPDATE, DELETE ON Temple TO temple_user;

1. Generalization Diagram: user as superclass for Customer and Admin.

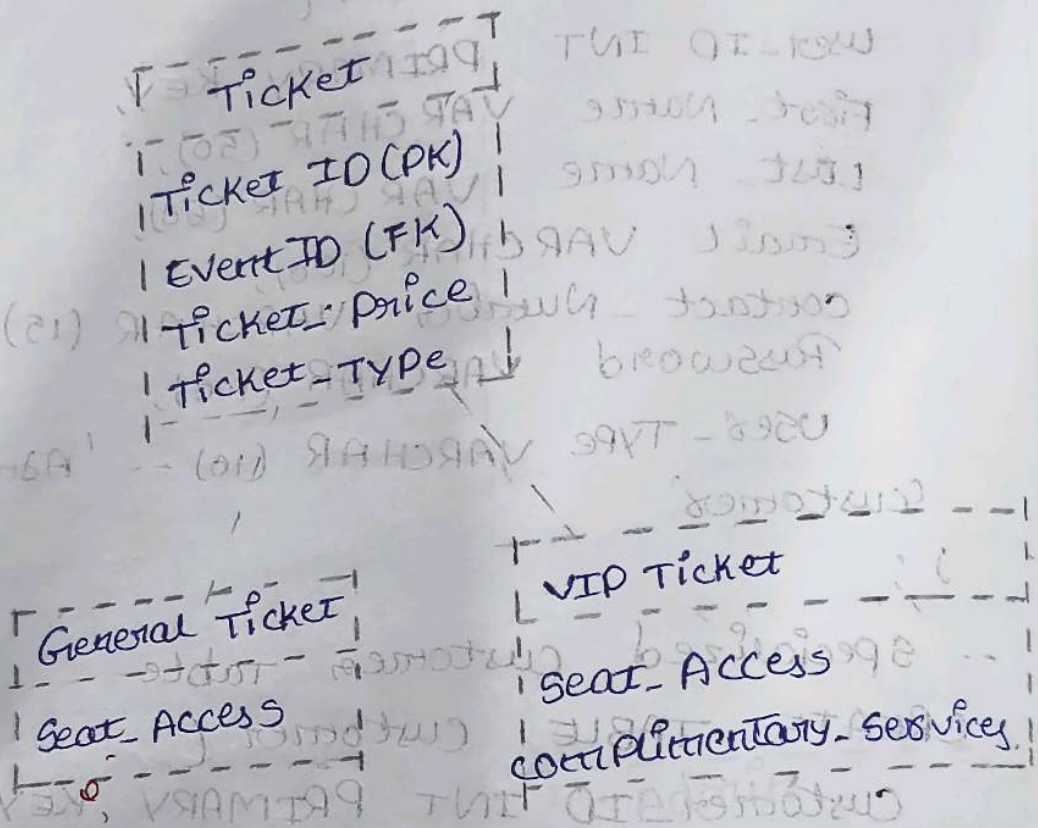


2.e Perform SQL Relations using DDL and DCL commands

DDL (Data Definition Language):

```
-- Create User Table (Generalization)
CREATE TABLE User (
    User_ID INT PRIMARY KEY,
    First_Name VARCHAR (50),
    Last_Name VARCHAR (50),
    Email VARCHAR (100),
    Contact_Number VARCHAR (15),
    Password VARCHAR (100),
    User_Type VARCHAR (10) -- 'Admin' or
'Customer'
);
-- Specialized Customer Table
CREATE TABLE Customer (
    Customer_ID INT PRIMARY KEY,
    User_ID INT,
    Age INT,
    Address VARCHAR (255),
    Payment_Info VARCHAR (255),
    FOREIGN KEY (User_ID)
    REFERENCES User (User_ID)
);
-- Temple Table
CREATE TABLE Temple (
    Temple_ID INT PRIMARY KEY,
    Name VARCHAR (100),
    Location VARCHAR (100),
    Description TEXT
);
```

2. Specialization Diagram: Ticket as superclass for General Ticket and VIP Ticket.



```
-- Event Table
CREATE TABLE Event (
    Event_ID INT PRIMARY KEY,
    Temple_ID INT,
    Name VARCHAR (100),
    start_Date DATE,
    End_Date DATE,
    FOREIGN KEY (Temple_ID)
REFERENCES Temple (Temple_ID)
);
-- Ticket Table with Specialization handled
 By ticket_type attribute
CREATE TABLE Ticket (
    Ticket_ID INT PRIMARY KEY,
    Event_ID INT,
    Ticket_type VARCHAR (20), --
'General', 'VIP'
    Ticket_Price DECIMAL (10, 2),
    FOREIGN KEY (Event_ID)
REFERENCES Event (Event_ID)
);
-- Booking Table (surplus relation for
many-to-many)
CREATE TABLE Booking (
    Booking_ID INT PRIMARY KEY,
    Customer_ID INT,
    TICket_ID INT,
    Booking_Date DATE
    Seat_Number INT
    FOREIGN KEY.
```

```
(customer_ID) REFERENCES
  customer (customer_ID),
      FOREIGN KEY (TICKET_ID)
  REFERENCES Ticket (TICKET_ID)
);
```

VEL TECH - CSE	
EX NO	9
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	13+5=18
SIGN WITH DATE.	

31/07/25

Result:-

Thus, the Hierarchical and Network models have been successfully created for the temple Ticket online Booking management System, with appropriate inheritance, relationship specificity, domain constraints, renamed relations, and SQL implementation using DDL and DCL commands.

7/8/25

Task 3

using clauses, operators and functions in queri

Aim: To Perform query Processing on a Temple Ticket online Booking management system for different retrieval results of queries using DML, DRL operations with aggregate functions, date functions, string functions, set clauses, and operators.

Temple

TempleID	Name	Location	Contact_No
T1D01	Tirumala Venkateswara	Tirupati	9876543210
T1D02	Meenakshi Amman	Madurai	9867543210
T1D03	Kashi Vishwanath	Varanasi	9856543210
T1D04	Jagannath Temple	Puri	9846543210
T1D05	Golden Temple	Amritsar	9836543210

Visitor

VisitorID	FName	LName	Age	Email	Contact_No
V001	Anil	Kumar	30	anilk@gmail.com	9123456780
V002	Akash	Reddy	25	akashr@gmail.com	9123456781
V003	Priya	Sharma	28	priyas@gmail.com	9123456782
V004	Aarav	Patel	35	aaravp@gmail.com	9123456783
V005	Sneha	Rao	22	snehar@gmail.com	9123456784

Booking

Booking ID	Temple ID	Visitor ID	Booking_Date	Ticket_Type	Amount
B001	TID01	V001	2024-06-15	VIP	500
B002	TID02	V002	2024-06-16	General	100
B003	TID01	V003	2024-06-17	General	100
B004	TID03	V004	2024-06-18	VIP	700
B005	TID01	V005	2024-06-19	Special	300

Priest

Priest ID	F Name	L Name	Age	Temple ID	Email	contact-No
P001	Ramesh	Iyer	50	TID01	Ramesh@gmail.com	9998887771
P002	Suresh	Sharma	45	TID02	suresh@gmail.com	9998887772
P003	Manish	Das	40	TID03	manish@gmail.com	9998887773

2. Retsieve details of visitors whose first name starts with 'A'

SQL

```
select *
FROM visitor
WHERE FName LIKE 'A%';
```

Result:

Visitor ID	F Name	L Name	Age	Email	contact NO
V002	Akash	Reddy	25	akash@gmail.com	9123456781
V004	Aarav	Patel	35	AaravP@gmail.com	9123456783

3. Add a column for 'special_seva' in Booking table.

SQL

```
ALTER TABLE Booking ADD special_seva VARCHAR(50);
```

Result:

Table altered Successfully.

4. Count the number of VIP ticket bookings.

```
sql
SELECT COUNT(*)
FROM Booking
WHERE Ticket_Type = 'VIP';
```

Result:

count(*)
2

5. Display temple details for Temple IDs 'TID01', 'TID03', and 'TID05'

```
sql
SELECT *
FROM Temple
WHERE Temple ID IN ('TID01', 'TID03', 'TID05');
```

Result:

Temple ID	Name	location	contact No
TID01	Tirumala Venkateswara	Tirupati	9876543210
TID03	Kashi Vishwanath	Varanasi	9866543210
TID05	Golden Temple	Amritsar	9836543210

6. Select visitor ID and names of visitors who booked 'special' tickets.

```
sql
SELECT visitor ID, FName, LName
FROM visitor
WHERE visitor ID IN (
    select visitor ID FROM Booking WHERE Ticket_Type
        = 'special'
);
```

Result:

visitor ID	F Name	L Name.
V005	Sneha	Rao.

7. Find the PriestID who have not been assigned any temple.

Sql

```
SELECT PriestID
 FROM priest
 WHERE TempleID IS NULL;
```

Result;

PriestID

(No results if all Priests are assigned)

VEL TECH - CSE	
EX NO	3
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	18
SIGN WITH DATE	

Result:-

Thus, Query Processing for Temple Ticket Online Booking management system using clauses, operators and functions was successfully performed.

14/8/24

Task 4

using functions in queries and writing sub queries:

Aim: To Perform advanced query Processing and test its heuristics by designing optimal correlated and nested subqueries such as finding summary statistics in the Temple Ticket online Booking management system.

4.1 To retrieve all temple details, including the count of bookings for each temple.

SQL

```
SELECT t.TempleID,
       t.Name AS TempleName,
       t.Location,
       t.Contact_No,
       COUNT(b.BookingID) AS TotalBookings
FROM Temple t
LEFT JOIN Booking b
ON t.TempleID = b.TempleID
GROUP BY t.templeID, t.Name, t.Location, t.contact_no;
```

Output

TempleID	TempleName	Location	Contact_No	TotalBookings
TIDO1	Tirumala venkateswara	Tirupati	9876543210	3
TIDO2	meenakshi Amman	Madurai	9867543210	1
TIDO3	kashi vishwanath	varanasi	9856543210	1
TIDO4	Jagannath Temple	Puri	9846543210	0
TIDO5	Golden Temple	Amritsar	9836543210	0

4.2: To retrieve the total number of 'Special' seva bookings in a temple_wise manner.

SqL

```
SELECT t.Name AS templeName,
       COUNT(*) AS Total Special Bookings
FROM Temple t
JOIN Booking b
ON t.TempleID=b.TempleID
WHERE b.Ticket_Type='special'
GROUP BY t.Name;
```

Output

Temple Name	Total Special Bookings.
Tirumala venkateswara	1

4.3: To retrieve the ~~total~~ number of 'special details of temples where bookings include 'VIP' tickets.

SqL

```
SELECT *
FROM Temple
WHERE TempleID IN(
      SELECT TempleID
      FROM Booking
      WHERE Ticket_Type='VIP'
   );
```

Output:

TempleID	Name	Location	contact_No
TID01	Tirumala venkateswara	Tirupati	9876543210
TID03	Kashi vishwanath	varanasi	9856543210

4.4: To retrieve visitors and booking details of visitors who are above 25 years old.

SQL

```
SELECT V.Visitor ID,
       V.FName AS visitor Name,
       V.Age,
       B.BookingID,
       B.Booking_Date,
       B.Ticket_Type,
       B.Amount
FROM visitor V
JOIN Booking b
 ON V.VisitorID = B.visitorID
  WHERE V.Age > 25;
```

Output :-

Visitor-ID	Visitor Name	Age	BookingID	Booking-Date	Ticket-Type	Amount
V001	Anil	30	B001	2024-06-15	VIP	500
V003	Priya	28	B003	2024-06-17	General	100
V004	Aarav	35	B004	2024-06-18	VIP	700

4.5: To retrieve the details of temples with no bookings.

SQL

```
SELECT *
FROM Temple
WHERE TempleID NOT IN (
      SELECT TempleID
      FROM Booking
);
```

Output:

TempleID	Name	Location	Contact-No
TID04	Jagannath Temple	Puri	9846543210
TID05	Golden Temple	Amritsar	9836543210

4.6: To retrieve the templeid, temple name, and visitor name for a particular visitorid given.

Sql

```
SELECT t.TempleID,
       t.Name AS TempleName,
       V.FName AS visitor Name
FROM Temple t
JOIN Booking b
 ON t.TempleID = b.TempleID
 JOIN visitor V
 ON b.visitor ID = V.visitor ID
WHERE V.visitorID = 'v005';
```

Output:

TempleID	TempleName	visitorName.
TID01	Tirumala venkateswara	Sneha.

14/8/25

Result:- Thus, the queries using functions, joins and nested subqueries were successfully executed for the Temple Ticket online Booking management system.

21/8/25

Task 5

Writing join queries, equivalent, and/or recursive queries:

Aim: To Perform advanced query Processing and test its heuristics using join queries, equivalent queries, and recursive queries for the Temple Ticket online Booking Management System, which manages devotees, Priests, Poojas, tickets, and booking details.

Queries:

5.1 To retrieve all temples and their Poojas

Sql

```
SELECT t.Name AS Temple, P.Pooja Name, P.Price
FROM Temple t
JOIN Pooja ON t.TempleID = P.TempleID;
```

5.2 To list all bookings along with the devotee and Pooja details.

Sql

```
SELECT b.BookingID, d.FName AS Devotee, P.PoojaName,
       b.BookingDate, b.SlotTime, b.Status
FROM Booking b
JOIN Devotee d ON b.DevoteeID = d.DevoteeID
JOIN Pooja P ON b.poojaID = P.PoojaID.
```

5.3 Count the number of bookings made by each devotee

Sql

```
SELECT d.FName AS Devotee, COUNT(b.BookingID) AS
FROM Devotee d                                  Total Bookings
LEFT JOIN Booking b ON d.DevoteeID = b.DevoteeID
GROUP BY d.FName.
```

Output:

Temple	Pooja Name
Tirupati Temple	Suprabhata Seva
Tirupati Temple	Archana
Kanchi Temple	Abhishekam
Madurai Temple	Special Darshan

output:

BookingID	Devotee	PoojaName	Bookingdate	SlotTime	Slot
B101	Rajesh	Suprabhata Seva	22-jun-23	06:00 AM	confirm
B102	Meena	Archana	22-jun-23	9:00 AM	confirm
B103	Arjun	Abhishekam	23-jun-23	7:30 AM	cancel
B104	Kavitha	Special Darshan	23-jun-23	8:30 AM	confirm
B105	Suresh	Abhishekam	24-jun-23	6:30 AM	confirm

Out Put

Devotee	Total Bookings.
Rajesh	1
Meena	1
Arjun	1
Kavitha	1
Suresh	1

5.4 To find all devotees who booked 'Abhishekam'

Sql

```
SELECT d.DevoteeID, d.FName, d.MobileNO
FROM Devotee d
JOIN Booking b ON d.DevoteeID = b.DevoteeID
JOIN Pooja p ON b.PoojaID = p.PoojaID
WHERE P.PoojaName = 'Abhishekam'.
```

5.5 To retrieve all Poojas and the number of times they were booked.

Sql

```
SELECT P.PoojaName, COUNT(b.BookingID) AS
FROM Pooja P                          Total Bookings
LEFT JOIN Booking b ON P.PoojaID = b.PoojaID
GROUP BY P.PoojaName.
```

5.6 To retrive the total number of cancelled booking temple wise.

Sql

```
SELECT t.Name AS Temple, COUNT(b.BookingID)
FROM Temple t                    AS cancelled Bookings
JOIN Booking b ON.t.TempleID = b.TempleID
WHERE b.status = 'cancelled'
GROUP BY t.Name;
```

5.7 To retrive temple details where booking were successful

Sql

```
SELECT DISTINCT t.TempleID, t.Name, t.locati
FROM Temple t                        t.Head Priest
JOIN Booking b ON t.TempleID = b.TempleID
WHERE b.status = 'confirmed'.
```

Output

DevoteeID	FName	Mobile No
D103	Arjun	9876543210
D105	Suresh	8765432190

Output

Pooja Name	Total Bookings
Suprabhata Seva	1
Archana	1
Abhishekam	2
Special Darshan	1

Output

Temple	cancelled Bookings
Kanchi Temple	1

Output

TempleID	Name	Location	Head priest
T001	Tirupati Temple	Tirupati	Ramanujacharya
T002	Kanchi Temple	Kanchipuram	Narayan Das
T003	Madurai Temple	Madurai	Subramaniam

Output:

FName	Age	slot time	BookingID	Booking date	Status
Suresh	55	6:30 AM	B105	24-Jun-23	confirmed

Output

TempleID	Name	Location	Head Priest
T004	Rameshwaram Temple	Rameshwaram	Vishnu Devan

Output:

Temple	Pooja Name	Devotee	Booking date	Slot Time
Tirupati Temple	Archana	Meena	22-Jun-23	9:00 AM

5.8 To retrieve devotee and booking details for devotees above 50 years old.

```
Sql
SELECT d.FName, d.Age, b.BookingID, b.BookingDate,
FROM Devotee d                    b.slottime, b.status
JOIN Booking b ON d.DevoteeID = b.DevoteeID
WHERE d.Age > 50;
```

5.9 To retrieve the details of temples where no Pooja bookings are done.

```
Sql.
SELECT *
FROM Temple
WHERE TempleID NOT IN (SELECT TempleID
                       FROM Booking);
```

5.10 To retrieve temple, Pooja and devotee details for a given Booking ID

```
Sql
SELECT t.Name AS temple, P.PoojaName, d.FName AS
       Devotee, b.BookingDate, b.slottime
FROM Booking b
JOIN Temple t ON b.TempleID = t.TEMPLEID
JOIN Pooja P ON b.PoojaID = P.PoojaID
JOIN Devotee d ON b.DevoteeID = d.DevoteeID
where b.BookingID = 'B102';
```

VEL TECH - CSE

EX NO	5
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	13+5=18
SIGN WITH DATE	

21/8/25

Results:- Thus, the temple ticket online Booking management System queries using Join Queries, equivalent queries, and recursive queries were executed successfully to manage temples, devotees, Priests, Poojas and bookings

28/8/25

Task-6

Procedures, Functions, and Loop using PSL/SQL on Number theory and Business scenarios

Aim:- To write and execute PSL/SQL Procedures, functions, and loops on Number theory (eg, Prime number, Factorial) and bussiness scenarios (e.g, salary calculation, Discount calculation).

Software used: Oracle SQL/PL/SQL

Algorithm:-

(A) Number theory Example:- prime number check using Procedure

1. Start
2. Read a number n
3. Initialize counter.
4. Use loop to check divisibility from 2 to n/2
5. If divisible ⇒ Not prime, else Prime
6. Display result.

(B) Business scenario Example - Salary Bonus calculation using Function.

1. Start
2. Create a function that accepts emp_salary.
3. If salary < 20000 ⇒ 20% bonus. If salary between 20000 - 50000 ⇒ 10% bonus
   Else ⇒ 5% bonus.
4. Return final salary with bonus
5. End.

Programs :-

A) Number Theory - Prime Number check (Procedure with LOOP)

```
SET SERVER OUTPUT ON;
CREATE OR REPLACE PROCEDURE
check_Prime (n IN Number ) is
    flag BOOLEAN := TRUE;
BEGIN
  IF n<=1 THEN
    DBMS_OUTPUT.PUT_LINE(n||' is NOT prime.');
 ELSE
  FOR i in 2..n/2 LOOP
    if MOD(n,i)=0 then
          flag:=FALSE;
          EXIT;
        END IF;
      END LOOP;
    IF flag THEN
      DBMS_OUTPUT.PUT_LINE(n||" is Prime.");
    ELSE
     DBMS_OUTPUT.PUT_LINE(n||' is NOT Prime');
     END IF;
   END IF;
END;
/
-- Execution
BEGIN
    Check_Prime(29);
    Check_Prime(20);
END;
/
```

Output:

For Prime Number check:

29 is Prime

20 is NOT Prime

For Bonus calculation:

Final Salary with Bonus: 21600

Final Salary with Bonus: 33000

(B) Business scenario- Employee Bonus calculation (Functio

```
SET SERVER OUTPUT ON;
CREATE OR REPLACE FUNCTION
calc_bonus (salary Number)
RETURN NUMBER IS
   bonus NUMBER;
BEGIN
   IF salary < 20000 THEN
      bonus := salary * 0.20;
   ELSIF salary BETWEEN 20000 AND 50000 THEN
      bonus := salary *0.10;
   ELSE
      bonus:= salary * 0.05;
   END IF;
   RETURN (salary + bonus);
END;
/
-- Execution
DECLARE
   final_sal NUMBER;
BEGIN
   final_sal := calc bonus (18000);
   DBMS_OUTPUT_PUT_LINE ('Final salary with
                          Bonus :' || final_sal);
   final_sal := calc bonus (30000);
   DBMS_OUTPUT_PUT_LINE ('Final salary with
                          Bonus :' || final_sal);
END;
/
```

VEL TECH-SET	
EX NO	6
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	+5
TOTAL (20)	13+5=18

28/8

Result: Thus, PL/SQL Procedures, functions, and loops were successfully implemented for Number Theory and Business scenarios

Task - 7

Triggers, Views and Exceptions

Aim: To conduct events, views, and exception handling on CRUD operations for the Temple Ticket online Booking management system.

a) Trigger

Requirement: when a new booking is inserted into the Booking table. automatically insert a corresponding record into the Ticket table with default status as 'ACTIVE'.

Sql

```
CREATE OR REPLACE TRIGGER
 trg_insert_ticket
 AFTER INSERT ON Booking
 FOR EACH ROW
 BEGIN
   INSERT INTO Ticket (TicketID, BookingID,
    DevoteeName, Age, Gender, QRcode, status)
    VALUES(
      seq_ticket.NEXTVAL, --assuming a
    sequence for ticket IDs
      :NEW.BookingID,
      'Unknown',      -- Default Place holder
      NULL,      -- age not given at booking time
      NULL, --gender not given at booking time
     SYS_GUID(), -- auto-generate QR code
     'ACTIVE' -- default ticket status
    );
 END;
 /
```

1. output

Trigger created

2. output

view created

3. output

Procedures created

BookingId	confirmation No	Visitor Name	Temple Name	Booking date
401	BK2005105	John	Brihadeeswara Temple	05-Dec-25
355	BK206106	Siva	Meenakshi Temple	20-Nov-25
500	BK207107	Priya	Golden Temple	1-Oct-25
300	BK208108	Ganesh	Somanath Temple	15-Sep-25

b) View

Requirement: create a view to display booking details along with temple and slot information.

Sql

```
CREATE OR REPLACE VIEW BookingDetails view
AS
SELECT
    b.BookingID,
    b.BookingRef,
    u.Name AS DevoteeName,
    t.Name AS TempleName,
    dt.Name AS DarshanType,
    s.StartTs AS SlotStart,
    s.EndTs AS SlotEnd,
    b.QTY,
    b.Amount,
    b.status
FROM Booking b
JOIN Users u ON b.UserID = u.UserID
JOIN Temple t ON b.TempleID = t.TempleID
JOIN Slot s ON b.SlotID = s.SlotID
JOIN DarshanType dt ON s.DarshanTypeID =
  dt.DarshanTypeID;
```

Sql

```
-- To query
SELECT * FROM BookingDetails view;
```

c) Non-Recursive PL/SQL Procedure

Requirement: Retrieve even-numbered BookingIDs for any given temple (similar to even PlayerIDs).

4. Output

1 row inserted
1 row inserted

5. output

Total bookings made today = 1

1 row updated

PL/SQL trigger completed successfully

6. output

Booking id	Name	Booking date	Time slot	No. of Persons	Total amount
305	Ravi	03-Sep-25	Morning	1	300

7. Output

Booking 1305 cancelled successfully

Sql

```
CREATE OR REPLACE PROCEDURE
Get Even Booking IDs For example(
    in_temple_id Number
    out_even_booking_ids OUT
SYS. ODCI NUMBER LIST
) AS
BEGIN
    out_even_booking_ids := SYS.ODCINUMBERLIST();
 FOR rec IN(
     SELECT Booking ID
     FROM Booking
     WHERE Temple ID = in_temple_id
         AND MOD (Booking ID, 2) = 0
    ) LOOP
      out_even_booking_ids. EXTEND;
 out_even_booking_ids (out_even_booking_ids.
        COUNT) := rec. Booking ID;
       END LOOP;
   END;
   /
Execution Block:
Sql
DECLARE
    temple_id NUMBER := 101;
    even_booking_ids. SYS. ODCI NUMBER LIST;
 BEGIN
    Get Even Booking IDs For Temple (temple_id,
  even_booking_ids);
    FOR i IN 1.. even _ids. COUNT. LOOP
      DBMS_OUTPUT. INE ('Even Booking ID:'
          || even_Booking_ids (i));
        temple_id .Number := 101;
        even_booking_ids SYS.ODCI Number List;
```

8. output

Error ORA-20001: Booking Id not found.

```
BEGIN
  Get Even Booking IDs For Temple (temple_id,
  even_booking_ids);
  FOR i IN 1.. even_booking_ids.COUNT Loop
     DBMS_OUTPUT.PUT_LINE ('Even Booking
        ID:' || even_booking_ids (i));
     END LOOP;
  END;
/
```

VEL TECH - CSE	
EX NO	7
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	+5
TOTAL (20)	5+5+5=20
SIGN WITH DATE	

Result: Thus, the Triggers, views, and Exceptions for the Temple Ticket online Booking management system were successfully implemented and verified.

TASK-8

CRUD operations in document database

Aim :- To design a document database for online Temple ticket booking management System using Mongoose (NPM) with Mongo DB and Perform CRUD Operations Create, read, Update, Delete tickets, temples, and user bookings.

Algorithm :-

Step 1 : Install MongoD community Edition

Step 2 : install Mongo shell (Mongosh)

Step 3 : Configure PATH Environment variable to include Mongosh binary.

Step 4 : confirm installation

```
Bash
mongosh --help
```

Step 5 : Start MongoDB server

```
Bash
C:/ Program Files/ MongoDB/ Server/ bin/ mongod.exe
```

Step 6 : Perform CRUD operations on Temple Ticket booking Database.

CRUD operations :-

1) i) Insert Temples :

db.Temples.insertOne({

Temple_ID : "TID01",

Name : "Meenakshi Amman Temple",

Location : "Madurai",

contact : 9876543210,

Timings : "6:00AM - 9:00PM"

});

ii) db.Temples.insertMany([

{Temple ID : "TID02", Name : "Ramanathaswamy Temple", Location : "Rameswaram", contact : 9765432109, Timings : "5:00 AM - 8:00 PM"},

{Temple ID : "TID03", Name : "Brihadeeswarar Temple", Location : "Thanjavur", contact : 9654321098, Timings : "7:00 AM - 8:30 PM"}

]);

Output :-

```
{
  "acknowledged" : true,
  "insertedId" :
      ObjectId("651d1c9e5ebbbte7993ad
}
```

Output :-

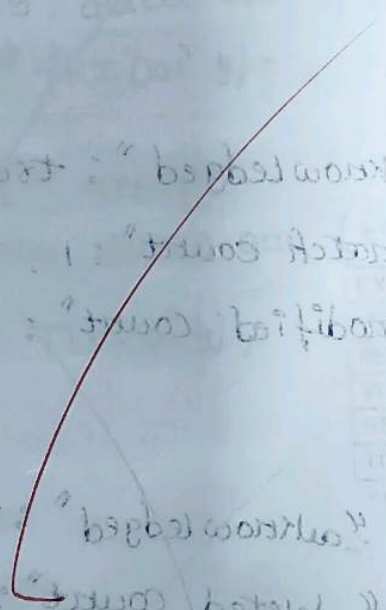
```
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("651d1cc36ebbfe7993adf911"
    ObjectId("651d1cc36ebbfe7993adf9
  ]
}
```

2) Insert ticket Booking

```
db.Tickets.insertOne({
    TicketID: "TK1D01",
    TempleID: "T1D01",
    userName: "John Doe",
    BookingDate: "2025-09-10",
    NumberofPersons: 3
});
```

3) Query All Temples

```
db.Temples.find().Pretty();
```



output:

{ "acknowledged": true,
"inserted Id":
objectId("651d1ce06e66bfe7993adf9

output:

{ "_id":
objectId("651d1c986e66bfe7993adf91c
"Temple Id": "TIDO1",
"Name": "Meenakshi Amman Temple"
"Location": "Madurai",
"contact": 9876543210,
"Timings": "6:00 AM - 9:00 PM"
}

"_id":
object_id("651d1ec36e66bfe7993a df911"
"Temple Id": "TIDO2"
"Name": "Ramanathaswamy Temple
"Location": "Rameswaram",
"contact": 9765432109,
"Timings": "5:00AM - 8:00 PM"
}

"_id":
objectId("651d1cc36e66bfe7993adf91a
"Temple ID": "TIDO3",
"Name": "Brihadeeswara Temple",
"Location": "Thanjavur",
"contact": 9654321698,
"Timings": "7:00AM - 8:30 PM"
}

output :-

{ "_id":
ObjectId("6518ce06e66fe7993adf913"),
"TicketID": "TKID01",
"TempleID": "TID01",
"UserName": "John Doe",
"BookingDate": "2025-09-10",
"NumberofPersons": 3
}

output :-

{
"acknowledged": true,
"matchcount": 1,
"modifiedcount": 1
}

output :- { "acknowledged": true,
"deletedcount": 1
}

output :-

{ "acknowledged": true
"deletedcount": 1
}

4) Query ticket Booking by userName

```
db.Tickets.find({userName: "John Doe"}).pretty();
```

5) update Temple contact

```
db.Temples.updateOne(
  {TempleID: "TID01"},
  {set: {Contact: 9998887776}}
);
```

6) Delete a ticket Booking

```
db.Tickets.deleteOne({TicketID: "TKID01"});
```

7) Delete a Temple

```
db.Temples.deleteOne({~~TempleID~~: "TID03"});
```

VEL TECH - CSE	
EX NO	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	14+5=19
SIGN WITH DATE	11.09.24

Result:- Thus, using MongoDB with Mongoos, we successfully designed a document database for an online Temple Ticket Booking Management System and Performed CRUD operations.

18/9/25

Task-9

CRUD operations in Graph databases

Aim: To Perform crud operations like creating, querying, updating, and deleting on graph spaces for an online temple ticket Booking Management system.

Step 1: Set up Neoj AuraDB

* click start free → continue with google → open
* copy the downLoaded Password files into and login to the Aura console.
* start Executing queries in the Neoj Browser.

Create nodes with Properties

Each node represents an Entity like devotee, Temple and Booking.

Create a Devotee Node

```
CREATE (d: Devotee & DevoteeID: 'D001',
  Name : 'Raj kumar',
  Age : 30,
  Phone : '9876543210',
  Return D
```

Create a Temple Node

```
CREATE (t : temple & Temple ID : 'T001',
  Name : 'Sri Venkateshwara Temple',
  Location : 'Tirupati'
  Return t.
```

Output

d.devoteeID	d. Name	d. Age	d. Phon
D001	Rajkumar	30	9876

Output

t. TempleID	t. Name	t. Location	t. cap
T007	Sri venkatesh-wara Temple	Tirupati	5000

→ Create a Booking Node

```
CREATE (b: Booking & Booking ID: 'B001',
Devotee ID: 'D001',
Temple ID: 'T001',
Return b.
```

Create Relationships

→ Devotee Makes Booking

```
MATCH (d: Devotee {Devotee ID: 'D001'}).
CREATE (d)-[:MADE]→(b)
RETURN d, b.
```

→ Booking linked to temple.

```
MATCH (b: Booking {Booking ID: 'B001'})
CREATE (b)-[:FOR]→(t)
RETURN b, t.
```

Display All Nodes

```
MATCH (n) Return n
```

Retrieve Particular Booking Details.

```
MATCH (b: Booking {Booking ID: 'B001'})
RETURN b.
```

Update Particular Devotee Details.

```
MATCH (d: Devotee {Devotee ID: 'D001'})
SET d.phone = '9876543210', d.Age = 31
RETURN d.
```

Output

b.BookingId	d.devoteeId	b.TempleId
B001	D001	T001

output

* Show nodes and relationship (FOR) Booking to Temple.

DELETE Particular Booking

MATCH (b: Booking {BookingID: 'B001'})

DELETE b.

VEL TECH - CSE	
EX NO	9
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	4
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	18
SIGN WITH DATE	18.09.24

Result:- By following the above steps we successfully created, inserted, queries, updated, and deleted nodes and relationships in the temple ticket Booking management system using Neo4j Graph Database.

Task - 10

Normalizing databases using functional dependencies up to Third Normal Form

Aim:- To normalize the below relation and create the simplified tables with suitable constraints for the online Temple Ticket Booking Management system.

Given Relation:

Temple Booking (
Booking ID, User ID, U Name, U EMail, U contact, Temple ID, T Name, T Location, Priest ID, Priest Name, Priest contact, Ticket ID, Booking Date, Darshan Date, Timeslot, Ticket Type, Price, Payment ID, Payment Mode, Payment Status, Transaction Date)

Step (a): Apply Functional Dependencies ⇒ Normalize to INF

First, identify functional dependencies (FDS):

1. User ID ⇒ U Name, U EMail, U contact
2. Temple ID ⇒ T Name, T Location.
3. Priest ID ⇒ Priest Name, Priest contact
4. Ticket ID ⇒ Ticket Type, Price
5. Payment ID ⇒ Payment Mode, Payment status, Transaction Date.
6. Booking ID ⇒ User ID, Temple ID, priest ID, Ticket ID, Booking Date, Darshan Date, Timeslot, Payment ID.

Since all attributes already contain atomic values (no repeating groups, no multivated attributes), the relation is already in INF.

Step (b): Normalize using FD+ and α+ candidate keys:

* Booking ID (main unique identifier of a booking)

From α+ (closure of candidate keys):

* Booking ID* = {All attributes} ⇒ confirms Booking ID is a candidate key.
* User ID, Temple ID, Priest ID, Ticket ID, Payment ID also act as foreign keys with their own dependencies.

Step (c): Minimal cover / canonical cover

We reduce FDs to minimal form:

1. User ID ⇒ U Name, U Email, U contact
2. Temple ID ⇒ T Name, T Location
3. Priest ID ⇒ priest Name, Priest contact.
4. Ticket ID ⇒ Ticket Type, Price
5. Payment ID ⇒ Payment Mode, Payment status, Transaction Date
6. Booking ID ⇒ User ID, Temple ID, Priest ID, Ticket ID, Booking Date, Darshan Date, Timeslot, Payment ID.

Step (d): Normalize to 2NF

Conditions for 2NF:

* Must already be in 1NF.
* No Partial dependency

Here, Booking ID is the primary key so no Partial dependency exists.

The schema is in 2NF.

1NF Output Table:

Customer Name	product id	quantity
Alice	P1	2
Alice	P2	3
Bob	P3	7

Step (e): Normalize to 3NF

Condition for 3NF:

* Must already be in 2NF.
* No transitive dependencies

Check:

* Booking ID ⇒ UserID ⇒ UName (transitive dependency).
* BookingID ⇒ TempleID ⇒ TName, TLocation (transitive dependency).
* Booking ID ⇒ PriestID ⇒ Priest Name (Transitive dependency)
* Booking ID ⇒ TicketID ⇒ TicketType, price (Transitive dependency)
* Booking ID ⇒ PaymentID ⇒ PaymentMode, Payment status (transitive dependency).

Final simplified Tables in 3NF

1. User Table
   User (UserID [PK], UName, UEMail, Ucontact)
2. Temple Table
   Temple (TempleID [PK], TName, TLocation)
3. Priest Table
   Priest (PriestID [PK], Priest Name, Priest contact)
4. Ticket Table
   Ticket (TicketID [PK], TicketType, Price.
5. Payment Table.
   Payment (PaymentID [PK], PaymentMode, Payment status, Transaction Date)
6. Booking Table
   Booking (BookingID [PK], UserID [FK], TempleID [FK] Priest [FK], TicketID [FK], PaymentID [FK], Booking Date, Darshan Date, Time slot)

output

MiN = {

A ⇒ B,

B ⇒ C,

AB ⇒ D

D ⇒ A

}

output

Student id	Student Name
101	Alice
102	Bob

output

Student id (PK)

Student Name

Student ID	Student Name
101	Alice
102	Bob

output

1. student

Student ID	Student Name
101	Alice
102	Bob

2. course

course ID	course Name
C01	Math
C02	Physics

Constraints:

* Primary keys: BookingID, UserID, TempleID, PriestID, TicketID, PaymentID.
* Foreign keys:
  * Booking.UserID ⇒ User.UserID
  * Booking.TempleID ⇒ Temple.TempleID
  * Booking.PriestID ⇒ Priest.PriestID
  * Booking.TicketID ⇒ Ticket.TicketID
  * Booking.PaymentID ⇒ Payment.PaymentID.
* unique constraints: UEmail, UContact.
* Check constraints:
  * Price > 0
  * Payment status ∈ {'pending', 'completed', 'Failed'}
  * Time slot within valid darshan timings.

VEL TECH - CSE	
EX NO	10
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	+5
VIVA VOCE (5)	5
RECORD (5)	+5
TOTAL (20)	20
SIGN WITH DATE	11.MTF ✓ 29.09.25

Result: Thus, the given relation for the online Temple Ticket Booking Management system has been normalized into simplified tables up to Third Normal Form (3NF) with suitable constraints, reducing redudancy and improving data integrity.

9/10/25

TASK 11

Menu forms and reports.

Aim :

To design an online Temple Ticket Booking management System using oracle Forms, menus, and Report Builder. The system involves creating a user interface (UI) for tickets online, managing bookings, and generating reports related to ticket sales and temple visit.

Install Oracle Forms and Report Builder.

Ensure oracle Forms and Report Builder are installed on your development machine to build and test the application.

Design the Data Model:

In oracle Forms, define the data model that connects to the database schema for the temple ticket booking system. Use the Data Block wizard to create data blocks that represent key tables or views such as

* Temples
  Ticket Types
  Booking Details
  Users / Devotees
  Payment Information

Set up data blocks for all necessary data to manage ticket booking, user details, and Payments.

Create Menus:

Menus provide the navigation structure for the booking system application.

Steps to create menus in oracle Forms:

1. Open oracle Forms Builder.
2. Create a new menu form or use an existing one
3. Add menu items for each major function such as:
   - Book Tickets
   - View Booking History
   - Cancel Tickets
   - Manage Temples
   - Reports.
4. Define Menu hierarchy.
5. Assign triggers or procedures to handle menu item actions.
6. Compile and run the menu from to test the navigation flow.

Design Forms:

Forms are used to capture, display, and edit data related to ticket booking

Steps to design forms in oracle Forms:

1. Create a new form for each major component of the booking system such as
   - Ticket Booking Form
   - User Registration Form
   - Payment Processing Form
   - Booking cancellation Form.

2. Add form elements like text fields, buttons, and lists.
3. use the property palette to configure element properties and link data blocks to database tables.
4. Write PL/SQL code for business logic, such as:
   Validating booking dates
   Checking ticket availbility
   Processing payments.
5. Test the forms inside oracle Forms Builder for functionality and data accuracy.

Create Reports:
Reports provide summarized information about booking and temple visits.

1. Open oracle Report Builder.
2. Create a new report or use an existing template.
3. Define the data source for the report using queries or PL/SQL procedures.
   Example reports:
   Daily Ticket Sales Report
   Booking summary by Temple
   Revenue Report.
4. Design the report layout with headers.
5. Add parameters to allow filtering
6. Generate and preview the report to verify the data accuracy and presentation.

VEL TECH - CSE	
EX NO	11
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	19
SIGN WITH DATE	

9/10/24

Result :-

Thus, designing an online Temple Ticket Booking management system with Oracle Forms, menus, and Report Builder involves creating an interactive UI for ticket booking and managing operations, along with generating detailed reports. The system is successfully developed to facilitate online bookings and efficient Temple management.

16/10/25

Usecase 4:

Army Supply chain, Bill of Materials and Maintenance Cost Management The nation's armed forces, support more than one million soldiers and about 200,000 civilian staff. Each of these staff members relies on multiple pieces of equipment, from helicopters and armored vehicles to small arms and radios, to complete their missions. With maintenance, operation, and support costs of equipment representing as much as 80% of total lifecycle costs, it's imperative that the Defense ministry track and analyze equipment maintenance costs including changing historical data sources Dimension with more flexibility like graph databases and to be given richer analysis like forecast replacement parts with the location and climate, mean time to failure rates, logistics and requirement processes. Is answerable the vital "what-if" questions such as the cost of deploying certain forces and supporting equipment to a new war zone? Could the model perform multi-dimensional cost comparison and trend analysis? Will the solution promises how data management in unpredictable maintenance costs?

Team Members:

1.PUVVADI SATHVIK
2.DONGALA PRANATHI
3.MODEM BHANU TEJA
4.GAJULA VIJAYA LAKSHMI
5.JAKKALA MAHESH
6.KOTTE SOOMANATH REDDI
7.MANAM RAMATULASI
8.ALUGUBELLY BHANUPRASAD REDDY
9.THOTA BINDU
10.GANGISETTY YASASWINI
11.PALAGIRI HARIKA
12.ATTADA KESAVA VENKATA DURGA AJAY
13.BALAGANI PURNA AJAY
14.YANDRAPU SHYAM DHANUSH
15.TEELLA BHANU PRASAD
16.REDDI YOGESH KUMAR

Aim:

To design and implement a Database Management System that:

- Tracks military equipment and their Bill of Materials (BOM)
- Records and analyzes maintenance events and costs
- Predicts equipment/component failure using Mean Time Between Failures (MTBF)
- Supports location- and climate-based failure forecasting
- Answers "what-if" questions like cost analysis of equipment deployment in different regions
- Enables multi-dimensional analysis of trends and comparisons over time.\

Algorithm:

1. MTBF (Mean Time Between Failures)

Purpose: Predict reliability of components.

Formula:

$$\text{MTBF} = \frac{\text{Total Operating Hours}}{\text{Number of Failures}}$$

Steps:

- Track total operational time between maintenance events
- Count the number of failure events for each component
- Divide operational hours by failure count

---

2. Forecasting Maintenance Cost (Time-Series)

Purpose: Estimate future maintenance expenses based on past trends.

Model: Prophet or ARIMA (Time-Series Forecasting)

Inputs:

- Historical maintenance costs by month
- Region and climate
- Equipment type

Outputs:

- Forecasted cost for upcoming months

Queries:

STEP 1: Create Tables

```
CREATE TABLE equipment (
  equipment_id VARCHAR(10) PRIMARY KEY,
  name VARCHAR(100),
  type VARCHAR(50),
  manufacturer VARCHAR(100),
```

```
  purchase_date DATE,
  status VARCHAR(20)
);
CREATE TABLE component (
  component_id VARCHAR(10) PRIMARY KEY,
  name VARCHAR(100),
  type VARCHAR(50),
  parent_equipment_id VARCHAR(10),
  mtbf INT,  -- Mean Time Between Failures in hours
  cost DECIMAL(10,2),
  FOREIGN KEY (parent_equipment_id) REFERENCES equipment(equipment_id)
);
CREATE TABLE location (
  location_id VARCHAR(10) PRIMARY KEY,
  name VARCHAR(100),
  region VARCHAR(50),
  climate VARCHAR(50)
);
CREATE TABLE maintenance_log (
  log_id SERIAL PRIMARY KEY,
  equipment_id VARCHAR(10),
  component_id VARCHAR(10),
  date DATE,
  downtime_hours INT,
  cost DECIMAL(10,2),
  issue_description TEXT,
  FOREIGN KEY (equipment_id) REFERENCES equipment(equipment_id),
  FOREIGN KEY (component_id) REFERENCES component(component_id)
);
CREATE TABLE deployment (
  deployment_id SERIAL PRIMARY KEY,
  equipment_id VARCHAR(10),
  location_id VARCHAR(10),
  date_from DATE,
  date_to DATE,
  type VARCHAR(50),
  FOREIGN KEY (equipment_id) REFERENCES equipment(equipment_id),
  FOREIGN KEY (location_id) REFERENCES location(location_id)
);
```

STEP 2: Insert Sample Data

```
INSERT INTO equipment VALUES
('EQ001', 'Tank M1', 'Tank', 'General Dynamics', '2020-01-15', 'Active'),
('EQ002', 'Apache AH-64', 'Helicopter', 'Boeing', '2019-06-10', 'Active');
INSERT INTO component VALUES
('CMP001', 'Engine', 'Mechanical', 'EQ001', 500, 12000.00),
('CMP002', 'Tracks', 'Mechanical', 'EQ001', 300, 5000.00),
('CMP003', 'Rotor Blade', 'Mechanical', 'EQ002', 400, 7000.00),
('CMP004', 'Avionics Unit', 'Electrical', 'EQ002', 600, 15000.00);
INSERT INTO location VALUES
('LOC01', 'Kashmir Base', 'North', 'Cold'),
('LOC02', 'Rajasthan Post', 'West', 'Desert');
INSERT INTO maintenance_log (equipment_id, component_id, date, downtime_hours, cost,
issue_description) VALUES
('EQ001', 'CMP001', '2025-01-10', 10, 2000.00, 'Oil leakage'),
('EQ001', 'CMP002', '2025-03-15', 5, 1500.00, 'Track misalignment'),
('EQ002', 'CMP003', '2025-02-05', 12, 3000.00, 'Rotor damage'),
('EQ002', 'CMP004', '2025-03-01', 8, 4500.00, 'Avionics fault'),
('EQ001', 'CMP001', '2025-04-20', 15, 2500.00, 'Engine overheating');
INSERT INTO deployment (equipment_id, location_id, date_from, date_to, type) VALUES
('EQ001', 'LOC02', '2025-01-01', '2025-06-01', 'Combat'),
('EQ002', 'LOC01', '2025-01-01', '2025-06-01', 'Recon');
```

Step 3:Total Maintenance Cost per Equipment

```
SELECT equipment_id,
    SUM(cost) AS total_maintenance_cost
```

```
FROM maintenance_log
GROUP BY equipment_id;
```

Output:

equipment_id	total_maintenance_cost
EQ001	6000.00
EQ002	7500.00

Step 4:MTBF per Component

```
SELECT component_id,
    COUNT(*) AS failure_count,
    SUM(downtime_hours) AS total_hours,
    ROUND(SUM(downtime_hours) * 1.0 / COUNT(*), 2) AS mtbf
FROM maintenance_log
GROUP BY component_id;
```

Output:

component_id	failure_count	total_hours	mtbf
CMP001	2	25	12.50
CMP002	1	5	5.00
CMP003	1	12	12.00
CMP004	1	8	8.00

Step 5: Find Equipment Failing in Cold Climates

```
SELECT eq.equipment_id, COUNT(*) AS failure_count
FROM equipment eq
JOIN deployment dp ON eq.equipment_id = dp.equipment_id
```

```
JOIN location loc ON dp.location_id = loc.location_id
JOIN maintenance_log ml ON eq.equipment_id = ml.equipment_id
WHERE loc.climate = 'Cold'
GROUP BY eq.equipment_id;
```

equipment_id	failure_count
EQ002	2

Step6:Average Monthly Maintenance Cost for EQ001 in Desert Deployment

```
SELECT ROUND(AVG(cost), 2) AS avg_monthly_cost
FROM maintenance_log ml
JOIN deployment dp ON ml.equipment_id = dp.equipment_id
JOIN location loc ON dp.location_id = loc.location_id
WHERE ml.equipment_id = 'EQ001' AND loc.climate = 'Desert';
```

Output:

avg_monthly_cost
2000.00

Step 7: Total Cost Forecast for 3 Months in Desert

```
SELECT 2000.00 * 3 AS forecasted_cost;
```

Output:

forecasted_cost
6000.00

Step 8: Query for Historical Deployment Cost in Desert

```
SELECT AVG(ml.cost) AS avg_monthly_cost
```

```
FROM maintenance_log ml
JOIN deployment dp ON ml.equipment_id = dp.equipment_id
JOIN location loc ON dp.location_id = loc.location_id
WHERE loc.climate = 'Desert'
 AND dp.equipment_id = 'EQ-005';
```

Output:

avg_monthly_cost
$18,700

VeL TECH - CSE

EX NO	12
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	18
SIGN WITH DATE	

Result:-

Thus, The proposed database system was implemented using SQL to manage Army equipment, their components (Bill of Materials), maintenance logs, deployment data, and geographic location details. Once the schema was established and sample data inserted, several SQL queries were executed to derive actionable insights from the data.