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: G. Jagannadha Varma

VTU.NO: 27789

REG.NO: 24UECL0016

BRANCH: CSE(AIML)

YEAR/SEM: 2nd year / 3rd sem

SLOT: S10 L12.

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	: G. Jagannadha Varma	**BRANCH**	: CSE(AIML)
VTU NO.	: 27789	**REG.NO.**	: 24UECL0016
YEAR/SEM	: 2nd year/III sem	**SLOT NO.**	: S10L12

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).

31/10/25

SIGNATURE OF LAB HANDLING FACULTY **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**

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	5	18	31/7
3.	07/08/25	Developing queries with DML Single-row functions and operations	10	18	7/8
4.	14/08/25	Developing queries with DML Multi-row functions and operations	14	18	14/8
5.	21/08/25	Writing Join Queries, equivalent, and/or recursive queries	18	17	21/8
6.	28/08/25	Writing PL/SQL using Procedures, Function	23	17	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	31	18	18/9
10.	25/09/25	CRUD operations in Document databases	33	18	25/9
11.	09/10/25	CRUD operations in Graph databases	37	17	9/10
12.	16/10/25	Micro Project: Use Case 3 - Gift Coupon Application	40	18	16/10

Completed

Total Marks: 216/240

R.T. Thyagarajan

Signature of Faculty

Task-1 24/7/25

A. Temple ticket Online booking Management System

A temple ticket Online booking management System allows devotees to book tickets for temple visits. Special darshan, poojas, and other events Online, reducing physical queues and improving crowd Management. These Systems often include features like date and time slot selection, payment processing and the generation of tickets or passes.

Entity:-

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

Entity Set:-

A collection of Entities of the same type. for instance are 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:-

An association or interaction between two or more Entities. for example, a student might enroll in a course. Establishing a relationship between the "student" and "course" Entities.

Aim:-

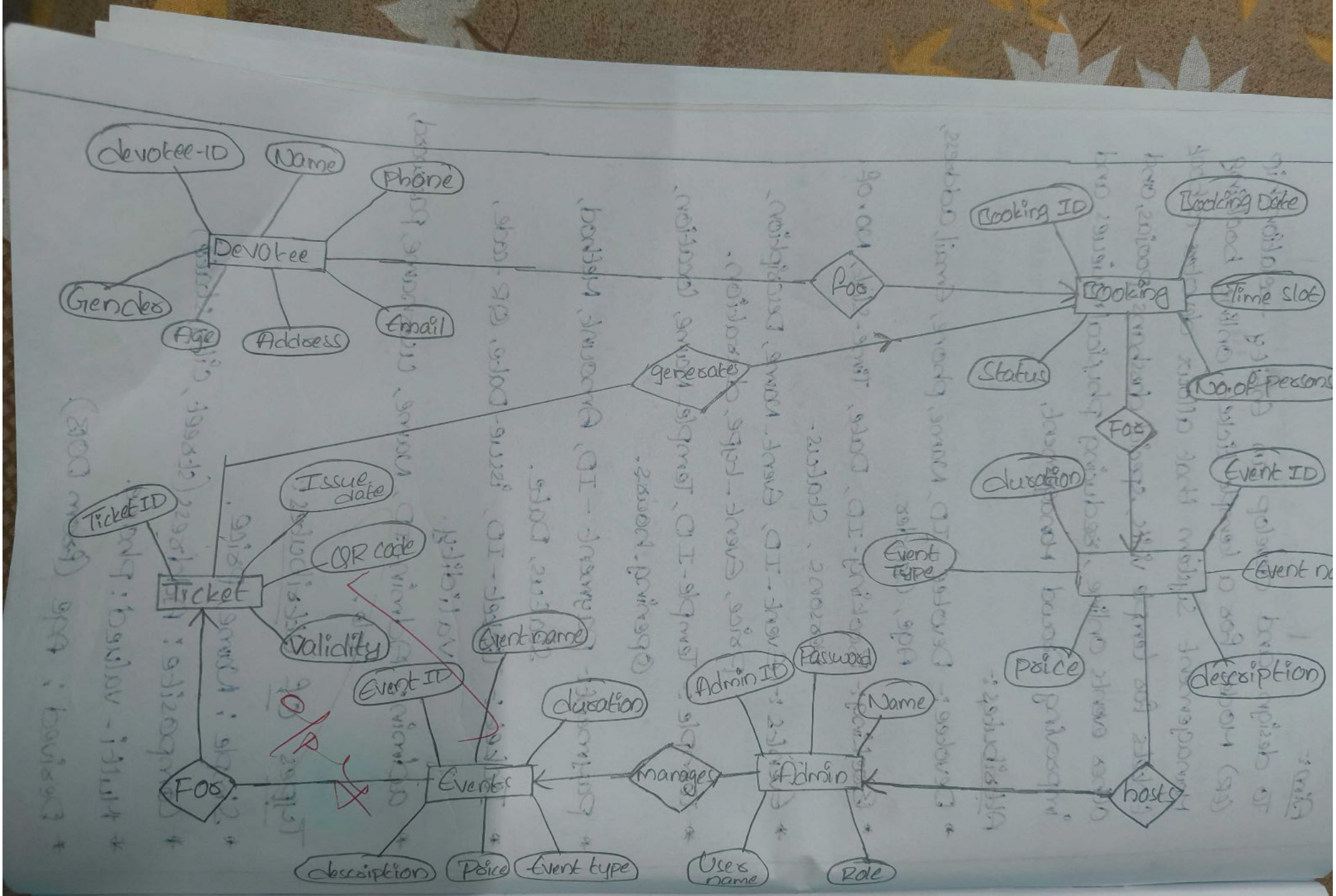
To design and develop an Entity-relationship (ER) Model for a temple ticket online booking Management System that allows devotees to book tickets for temple visits, special darshans, poojas, and other events online, reducing physical queues and improving crowd management.

Attributes:-

* Devotee:- Devotee-ID, Name, phone, Email, address, Age, Gender.
* Booking:- Booking-ID, Date, Time-slot, No. of. persons, Status-
* Events:- 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)


devotee-ID
Name
Phone
Devotee
Gender
Age
Address
Email
FOS
Booking
Booking ID
Booking Date
Time slot
Status
No.of persons
generates
FOS
Event
duration
Event ID
Event Type
Event n
Price
description
host
Ticket
Ticket ID
Issue date
QR code
Validity
FOS
Events
Event name
Event ID
duration
description
Price
Event type
manages
Admin
Admin ID
Password
Name
User name
Role

Relationships :-

* Devotee → Booking (Makes).
* Booking → event. (for).
* Booking → payment (has).
* Booking → Ticket (generates)
* Event → Temple (hosts).
* Admin → Event (Manages).

Relationship Types :-

* One-to-One : Booking → payment, Booking → Ticket.
* One-to many : Devotee → Booking.
* Many-to-One : Booking → Event.
* One-to-many : Temple → Events.
* 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)	18
SIGN WITH DATE	

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.

31/7/25

Task-2:- Generating design of other traditional database model.

Aim :- Creating Hierarchical 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 heirarchy has a heirarchy 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 1: 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).

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 ticket, and a ticket can be booked by many customers.

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.

2b. Check is a hierarchy/has-a hierarchy and perform generalization and /or specilization.

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 info:

General Ticket (attributes: price, general_access_area).

Vip Ticket (attributes: price, access_to_special_area, complimentary_services).

2c. 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).

2d. Rename the relations (tables).

Example renaming columns for clarity:

Sol

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;

2.e perform SQL Relations using DDL and DCL Commands.

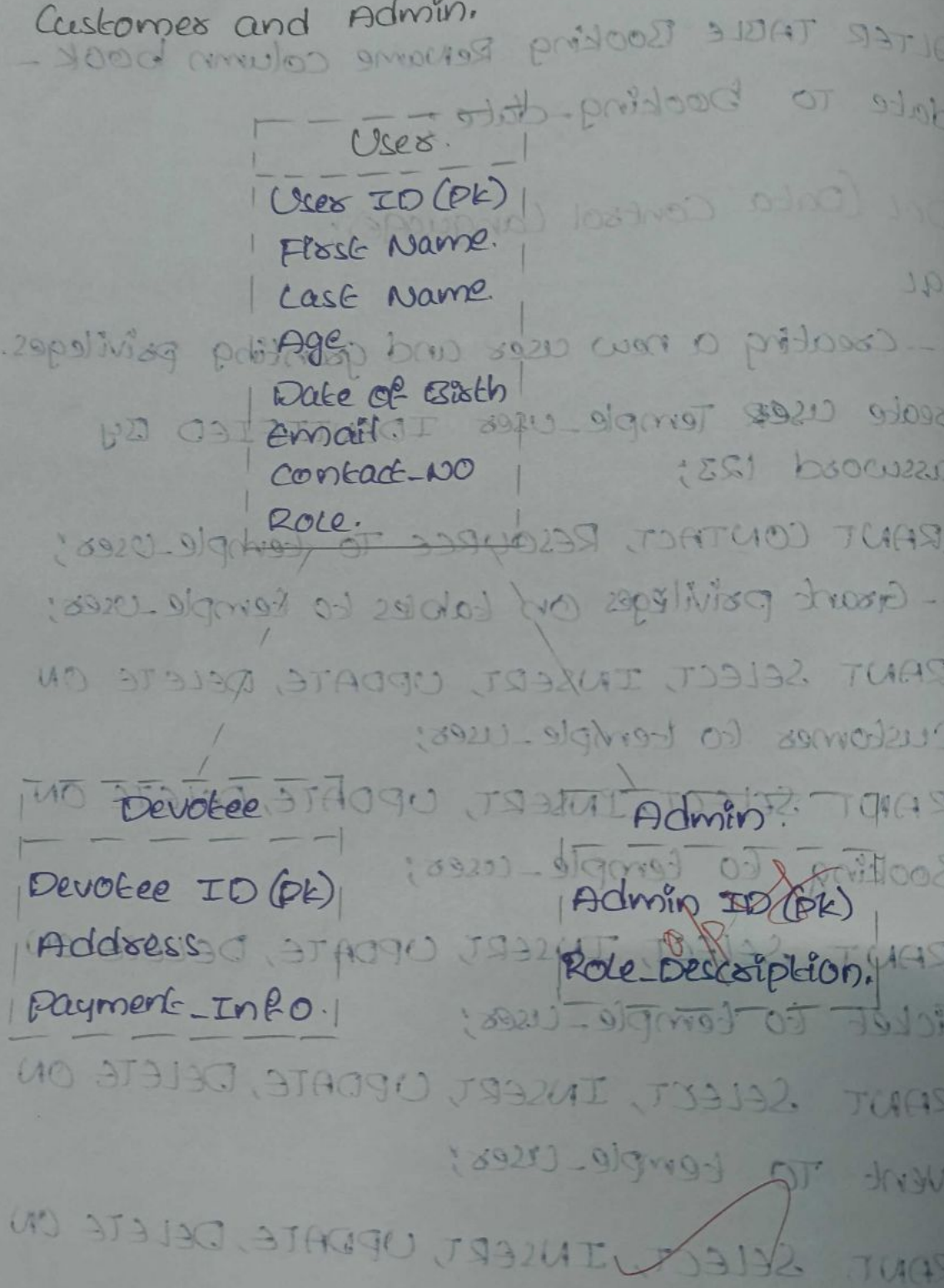
DDL (Data Defination Language):

-- Create User Table (Generalization)

CREATE TABLE User (.

User_ID INT PRIMARY KEY,

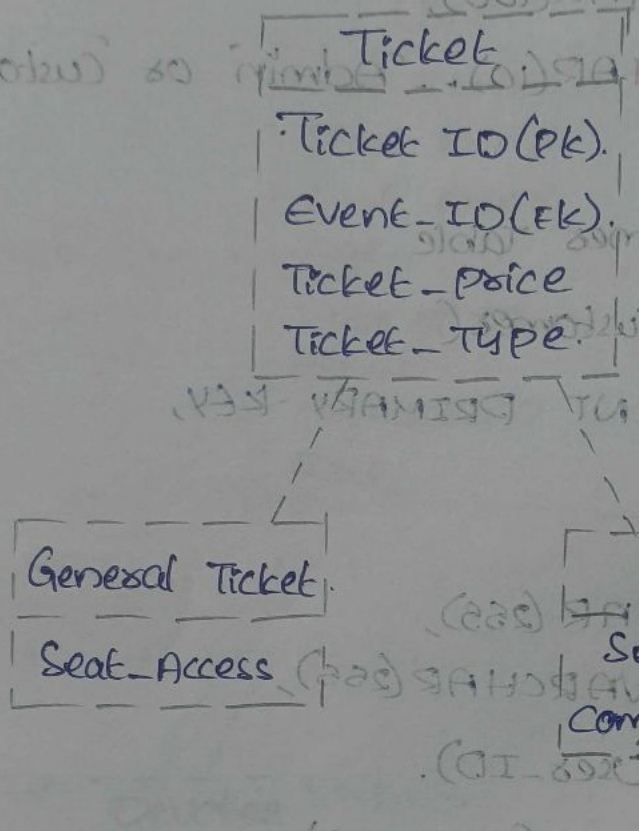
1. Generalization Diagram: Uses as superclass for Customer and Admin.



```
  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),
  FOREIGG 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.
);

-- Event Table.
CREATE TABLE Event(.
  Event_ID INT PRIMARY KEY,
  Temple_ID INT,
  Name. VARCHAR(100),
```

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



```
Start_Date DATE,
End_Date DATE,
FOREIGN KEY (Temple_ID),
REFERENCES Temple (Temple_ID),
);
-- Ticket Table with specialization handle 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	2
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	18
SIGN WITH DATE	

13+5=18

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 implementations using DDL and DCL commands.

7/8/25

Task-3:-

Using clauses, Operators and functions in queries.

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, data function, string function, set clauses, and operators.

Temple

TEMPLE ID	NAME	LOCATION	CONTACT_NO
TID01	Tirumala venkateshwara	Tirupati	9024276954
TID02	Meenakshi Amman	Madurai	7989222725
TID03	kashi vishwanath	Varanasi	7013982716.
TID04	Jagannath Temple	Puri	9701224404.
TID05	Golden Temple	Amritsar	9908616499.

Visitor

VisitorID	F.Name	L.Name	Age	Email	Contact.NO.
V001	Anil	Kumar	30	Anil@gmail.com	9014276954.
V002	Akash.	Reddy	25	Akash@gmail.com	9701224404
V003	Priya	Sharma	28	Priya@gmail.com	9908290010.
V004	Aarav	Patel	35	Aarav@gmail.com	7013982716
V005	Sneha.	Rao	22	Sneha@gmail.com.	7989222725.

Booking

Booking ID	Temple ID.	Visitor ID.	Booking_date	Ticket Type	Amount.
B001	TID01	V001	2024-6-15	VIP	500
B002	TID02	V002	2024-6-16	General	100.
B003	TID03	V003	2024-6-17	General	100
B004	TID04	V004	2024-6-18	VIP	500
B005.	TID05.	V005.	2024-6-19	General.	100.

Priest

Priest ID	F.Name	L.Name.	Age	Email	Contact_NO.
P001	Ramesh	Iyer	50	Ramesh@gmail.com	9014276954
P002	Suresh	Sharma	45	Suresh@gmail.com	9701229404
P003.	Manish	Das.	40	Manish@gmail.com.	7989222785.

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

```
SELECT *
FROM Visitor.
WHERE FNAME LIKE 'A%';
```

Result:-

Visitor ID.	F Name	L.Name	Age	Contact_NO
V002	Akash.	Reddy	25	9701224404
V004	Aarav	Patel.	35.	7013982716.

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

ALTER TABLE Booking ADD Special_seva VARCHAR(50);

Result:

Table Altered successfully.

4. Count the number of VIP ticket bookings.

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

Result:-

Count(*)

5. Display temple detail for Temple IDS 'T1D01', 'T1D03', and 'T1D04'.

SELECT *.
FROM TEMPLE.
WHERE TEMPLE ID. IN ('T1D01', 'T1D03' 'T1D04');

Result:-

Temple ID	Name.	Location.	Contact_NO.
T1D01.	Tirumala	Tirupati	9701224404.
T1D02	Kashi	Varanasi	7013952716
T1D03	Golden Temple	Amritsar.	7989222725.

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

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 priest ID of priests who have not been assigned any temple.

SELECT PRIEST ID.
~~FROM PRIEST.~~
FROM priest.
WHERE Temple ID is NULL;

Result:-
Priest ID.
(No Result if all priests are assigned).

VEL TECH - CSE	
EX NO 2	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	2
RECORD (5)	5
TOTAL (20)	17
SIGN WITH DATE	

Result:- thus, query processing for temple ticket. Online Booking Management System using clauses, Operation, and function was successfully performed

14/8/25

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 statics in Booking Management system.

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

```
SELECT t.Temple ID,
       t.Name AS TempleName,
       t.Location,
       t.Contact_NO,
   COUNT(b.Booking ID) AS Total Bookings.
FROM Temple t
LEFT JOIN Booking b.
ON.t.Temple ID=b.Temple ID.
Group By t.Temple ID, t.Name, t.Location, t.
Contact_NO;
```

Output:-

Temple ID	Temple Name.	Location.	Total Bookings
TID01	Tirumala	Tirupati	3
TID02	Meenakshi Amman	Madurai	1
TID03	Kashi	Varanasi	1
TID04	Jagannath temple	Puri	0
TID05.	Golden temple.	Amritsar.	0.

o/p

4.2: To retrieve the total number of 'special' seva booking in a temple_wise manner.

```
SELECT t.Name AS TempleName,
COUNT(*) AS TotalSpecialBookings.
FROM Temple t
JOIN Booking b.
ON t.TEMPLE ID=b.Temple ID.
WHERE b.Ticket_Type='Special'.
Group By t.Name;
```

Output:-

Temple Name.	Total Special Bookings.
Tirumala Venkateshwara	1.

4.3: To Retrieve the details of temples where booking include 'VIP' tickets.

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

Output:-

Temple ID	Name	Location	Contact_NO
TID01	Tirumala	Tirupati	9701224404
TID03	Kashi	Varanasi	7013982716.

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

```
SELECT *
FROM TEMPLE
WHERE Temple.ID NOT IN(
SELECT Temple ID.
FROM Booking
);
```

Output:-

Temple ID	Name	Location	Contact_NO
TID04	Jagannath Temple	Puri	9701224404.
TID05.	Golden Temple	Amritsar	9014276954.

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

```
SELECT t.Temple ID,
       t.Name AS Temple Name,
       V.F Name AS Visitor Name.
FROM Temple t.
JOIN Booking b.
ON t.Temple ID = b.Temple ID.
JOIN Visitor V.
ON b.Visitor ID = V.Visitor ID.
WHERE V.Visitor ID = 'V005';
```

Output:-

Temple ID.	Temple Name.	Visitor Name.
TID01	Tirumala	Sneha.

VEL TECH - CSE	
EX NO	4
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	13+5=18
SIGN WITH DATE	

14/8/25

Result:-

Thus, the queries using function, join and nested subqueries were successfully executed for the temple ticket Online Booking Management System.

21/8/25

Task-5 :- Writing Join Queries, equivalent, and Recursive Queries.

Aim :- To perform advanced query Processing and test heuristics using Optimal correlated and nested subqueries such as retrieving summary statistics and ticket booking details for the Online Temple Ticket Booking Management System.

5.1) To retrieve all temples and their available tickets,

```
SELECT tm.Name AS Temple, tk.Ticket ID, tk.Type,
                    tk.price.
FROM Temple tm
JOIN Ticket tk ON tm.Temple ID= tk.Temple ID;
```

5.2) To list all bookings along with temple name and devotee name.

```
SELECT b.Booking ID, d.DevoteeName, tm.Name AS
        TempleName, tk.Type, b.BookingDate.
FROM Booking b
JOIN Devotee d ON b.Devotee ID=d.Devotee ID.
JOIN Ticket tk ON b.Ticket ID=tk.Ticket ID.
JOIN Temple tm ON tk.Temple ID = tm.Temple ID;
```

S.1 Output

Temple.	Poojaname.	price.
Tirupathi	Subrabhatha sella	300.
Tirupathi	Archana	100
Kanchi	Abhishekham	500
Madhurai	special Darshan	250.

S.2 Output

Booking ID	Devotee.	Poojaname.	Book ~~date~~ time.	~~time~~ Date	status.
B101	Rajesh	Subrabhatha sella	6:00 AM.	22-7-23.	confirm
B102	Muna.	Archana	9:00 AM.	22-7-23	Confirm
B103	Arjun.	Abhishekham	7:30 AM.	23-7-23.	Confirm
B104	kavitha	special Darshan	8:30 AM.	23-7-23	Confirm
B105	Suresh.	Abhishekam	6:30 AM.	24-7-23.	Confirm.

S.3 Output

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

S.4 Output

Devotee id.	F.Name.	Mobile NO.
D103.	Arjun.	9876543210.
D105	Suresh.	8765432190.

5.3). To count the, number of booking made for each temple.

```
SELECT tm.Name As TempleName,
COUNT(b.Booking ID) As Total Bookings.
FROM Temple tm.
LEFT JOIN Ticket tk ON tm.Temple ID=tk.TempleID
LEFT JOIN Booking b ON tk.Ticket ID= b.Ticket ID.
GROUP BY tm.Name;
```

5.4) To find all devotees who booked tickets, for 'Tirupati Balaji'

```
SELECT d.Devotee ID, d.DevoteeName, d.contactNO,
        tm.Name As Temple.
FROM devotee d.
JOIN Booking b ON d.Devotee ID=b.DevoteeID.
JOIN Ticket tk ON b.Ticket ID = tk.Ticket ID.
JOIN Temple tm ON tk.TempleID= tm.Temple ID.
WHERE tm.Name = 'Tirupati Balaji';
```

5.5) To retrieve all devotee details including total tickets booked.

```
SELECT d.Devotee ID, d.DevoteeName, d.contactNO,
        COUNT(b.Booking ID) As Total Tickets.
FROM Devotee d.
LEFT JOIN Booking b ON d.Devotee ID=b.
                        DevoteeID.
```

S.5 Output

Pooja Name.	Total Bookings.
Suprabhata seva.	1
Archana.	1
Abhishekam.	2
Special Darshan.	1

S.6 Output

Temple.	Cancelled Bookings.
kanchi Temple.	1

S.7 Output

Temple ID.	Name.	Location.	Head Priest.
T001	Tirupathi	Tirupati	Ramanuja
T002.	kanchi	kanchipuram	Narayan Das.
T003.	Madurai	Madurai	Subramanian.

S.8 Output

F.Name	Age.	BookingID	BookingDate	Slot time	Status
Suresh	55	B105	24-7-23.	6:30AM.	Confirmed

5.6) To retrieve the total number of 'Special Darshan' tickets booked temple wise.

```
SELECT tm.Name As Temple,
    COUNT(b.BookingID) As special Darshan Count.
FROM Temple tm.
JOIN Ticket tk ON tm.TempleID = tk.TempleID
JOIN Booking b ON tk.TicketID = b.TicketID.
WHERE tk.Type = 'Special Darshan'
GROUP BY tm.Name;
```

5.7). To retrieve the temple details where tickets are still available (not fully booked).

```
SELECT tm.TempleID, tm.Name, tk.Type, tk.Price
FROM Temple tm.
WHERE tk.Available Seats >0;
```

5.8) To retrieve devotee and booking details for devotees above 40 years.

```
SELECT d.DevoteeID, d.Devotee Name, d.Age,
                b.Booking ID,
        b.BookingDate, tm.Name As Temple.
FROM Devotee d.
JOIN Booking b ON d.Devotee ID=b.devotee ID.
JOIN Ticket tm ON tk.Ticket ID=tk.Ticket ID.
WHERE d.Age>40;
```

5.9 Output

Temple ID.	Name	Location	Head Priest.
T004.	Rameshwaram Temple.	Rameshwa-ram.	Vishnu Devan.

5.10. Output:

Temple	Pooja Name.	Devotee.	Booking Date	Slot time
Tirupati	Archana.	Meena	22-7-23.	9:00AM.

5.9) To retrieve Temple details where no booking have been made.

```
SELECT * FROM Temple.
WHERE TempleID NOT IN(SELECT tk.TempleID.
                FROM Ticket Tk.
                JOIN Booking b ON tk.TicketID=
                    b.TicketID);
```

5.10) To retrieve temple ID, ticket ID, temple name and devotee for a particular booking ID given.

```
SELECT tm.TempleID, tk.TicketID, tm.Name AS
        TempleName, d.DevoteeName.
FROM Temple tm.
JOIN Ticket tk ON tm.TempleID=tk.TempleID.
JOIN Booking b ON tk.TicketID=b.TicketID
WHERE b.BookingID = 'B002';
```

VEL TECH - CSE

EX NO	5
PERFORMANCE (5)	4
RESULT AND ANALYSIS (5)	4
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	17
SIGN WITH DATE	04.09.25

Result:- Thus, the queries using Join queries, Nested queries, and equivalent queries were executed successfully for the online Temple Ticket BMS.

22/8/25

Task 6:- Procedures, Functions, and loops using PL/SQL on number theory and Business Scenarios.

Aim:-

To write and execute PL/SQL procedure functions and loops on number theory (eg:- prime number theory) and business scenarios (eg:- Salary calculation, Discount calculation).

Algorithm:-

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. It is divisible → Not prime, else prime.
6. Display Result.

(B) Business scenario example. Salary Bonus calculation using function.

1. Start
2. Create a function that accept emp-salary
3. If Salary < 2000 → 20 % bonus.
   If Salary between 20000 - 50000 → 10%. bonus else → 5% bonus.
4. Return final salary with bonus.
5. End.

Program:-

(A) Number theory - prime number check.
(Procedure with loop).

```
SET SERVEROUTPUT 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 loop
else
DBMS_OUTPUT_PUT LINE (n|| is not prime);
    END if;
    END IF;
    END;
    /
    check_prime (29);
    check_prime (20);
    END;
    /
```

Output:-

For the prime number check

20 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.
(function).

```
SET SERVER OUTPUT ON;
CREATE OR REPLACE function
      Return Number
   bonus number;
   Begin
  is Salary < 2000 then.
    bonus = salary * 0.20;
ELSE Salary Between 20000 AND 50000 then.
bonus = salary * 0.10;
    else
bonus = salary * 0.05;
    END IF
Return (salary * bonus);
    END;
    /
... Execution.
    Declare.
DBMS-Output-Put Line(Final salary with Bonus);
      // final sal);
final-sal = code = bonus(30000); DBMS-Output
Put line (Final salary with. Bonus: // final-sal);
      END;
      /
```

VEL TECH - CSE	
EX NO	6
PERFORMANCE (5)	4
RESULT AND ANALYSIS (5)	4
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	17
SIGN WITH DATE	04.05.24

Result:- Thus PL, SQL procedure functions and loops were successfully implemented for number theory and business scenarios.

4/9/25

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'.

```
CREATE OR REPLACE TRIGGER
trg_insert_ticket
AFTER INSERT ON Booking
FOR EACH ROW
BEGIN
    INSERT INTO Ticket (Ticket ID, Booking ID,
DevoteeName, Age, Gender, QRCode, Status).
    VALUES(
        Seq_ticket.NEXTVAL,
Sequence for ticket IDs
        :NEW.BookingID,
        'Unknown',
Placeholder
        NULL,
given at booking time
        NULL,
```

given at booking time.

SYS_GUID(),

auto-generate QR code

'ACTIVE'

ticket status

);

END;

/

b) View

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

```
CREATE OR REPLACE VIEW BookingDetailsView
AS
SELECT
    b.BookingID,
    b.BookingRef,
    u.Name AS DevoteeType,
    t.Name AS TempleType,
    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.User ID = u.User ID.
JOIN Temple t ON b.TempleID = t.TempleID.
JOIN Slot S ON b.SlotID = S.SlotID
JOIN DarshanType dt ON S.DarshanTypeID =
dt.DarshanTypeID;
```

```
-- TO query
SELECT * FROM BookingDetailsView;
```

c) Non-Recursive PL/SQL procedure.

Requirement : Retrieve even-numbered Booking IDs. for any given temple (similar to even playerIDs).

```
CREATE OR REPLACE PROCEDURE
GetEvenBookingIDsForTemple(.
    in_temple_id. NUMBER
    Out_even_booking_ids OUT
SYS.ODCINUMBERLIST
)AS
BEGIN
    Out_even_booking_ids: =
SYS.ODCINUMBERLIST(); -- Initialize the.
Collection.

    FOR rec IN(.
        SELECT BookingID.
        FROM Booking.
        WHERE TempleID = in_temple_id.
            AND MOD(BookingID, 2) = 0
    )LOOP.
        Out_even_booking_ids. EXTEND;
Out_even_booking_ids(Out_even_booking_ids.
COUNT):= rec.BookingID;
    END LOOP;
END;
/
```

```
DECLARE
    temple_id NUMBER := 101; -- Replace
with desired TempleID.
    even_booking_ids SYS.ODCINUMBERLIST;
BEGIN
    GetEvenBookingIDsForTemple(temple_id,
even_booking_ids);
    FOR i IN 1..even_booking_ids.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE('Even
BookingID: ' || 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)	18+5=20
SIGN WITH DATE	T.M.T 04.07.25

<u>Result</u>:- Thus, the triggers, views, and exceptions for the Temple Ticket Online Booking Management System were successfully implemented and verified.

11/9/25.

Task-8:- CRUD Operations in Document databases

Aim:- To perform Mongoose using NPM design on Mongo DB for online Temple Ticket BMS, designing a document DB and performing CRUD operations like creating, inserting, quer-ying, finding, updating, removing operations.

STEPS:-

1) Install Mongosh.
Download Mongo DB shell

2) Add the Mongo DB shell binary's location to you PATH.

* Open control panel → system and security. → system.
* Click Advanced system settings
* Click Environment variables
* Edit path and add the Mongo DB shell binary location.

4) Confirm PATH by typing in command prompt: Mongosh --help.

5) Open Mongo DB server.
C:\programfiles\Mongo DB\server\bin\Mongod.exe

6) Perform CRUD operations.

Output:-

2) {
"acknowledged": true,
"insertedId":
ObjectId("65ICf1726ebbfe7993adf999")
}

3) {
"_id": ObjectId("65ICf1726ebbfe999"),
"TicketID": "TID01",
"TempleName": "Meenakshi Amman Temple",
"Location": "Madurai",
"Booking Date": "2025-09-10",
"Ticket price": 200.

4) {
"acknowledged": true,
"insertedIds": [
ObjectId("65ICf1726ebbfe7993901"),
ObjectId("65ICf1726ebbfe7993902"),
ObjectId("65ICf1726ebbfe7993903"),
]
}

CRUD Operations for online Temple Ticket BMS.

1) Create Collection.

```
db.Create collection("Temple Tickets")
{"ok":1}
```

2) Insert a Single Document.

```
db.TempleTickets.insertone({
    TicketID:"TID01",
    TempleName:"Meenakshi Amman Temple",
    Location:"Madurai",
    DevoteeName:"Ramesh kumar",
    BookingDate:"2025-09-10",
    Ticketprice:200
})
```

3) Find one Document.

```
db.TempleTickets.find({TicketID:"TID01"})
```

4) Insert Multiple Documents.

```
db.TempleTickets.insertMany({
{
    TicketID:"TID02",
    TempleName:"Brihadeeshwara Temple",
    Location:"thanjavur",
    Ticketprice:150.
},
{
    TicketID:"TID03",
    TempleName:"Kapaleeshwarar Temple",
    Location:"chennai",
    Ticketprice:100
})
```

5) {
"TicketID": "T1001",
"TemplaeName": "Meenakshi Amman Temple",
"Location": "Madurai",
"Ticketprice": 200.
}

6) {"acknowledged": true, "matched count": 1,
"modified count": }

7) {"acknowledged": true, "deleted count": 1}.

7)

5) Find All Documents.

db.TempleTickets.find().Pretty().

6) Update a Document.

db.TempleTickets.updateOne(
{TicketID:"TID03"},
{$set:{TicketPrice:120,DevoteeName:"vignesh kumar"}}
)

7) Delete a Document.

db.TempleTickets.deleteone({TicketID:"TID02"}).

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	11/04/25

Result:- Thus CRUD Operation were successfully Performed using MongoDB with NPM/ Mongoose design.

18/9/25

Task-9:- Crud operations in Graph Database.

Aim:-

To perform CRUD operations like creating, inserting, querying, updating, and on graph Online Temple Ticket Booking.

Algorithm:-

Step 1:-

Open https://neo4j.com/cloud/platform/aura graph-data base/?ref=docs-gets-started-dropdown.

Step 2:-

Click start free → continue with google → Open.

Step 3:-

Copy the downloaded password file info and login to the Aura console.

Step 4:-

Start executing queries in the Neo4j Browser.

*Create a Devotee Node

CREATE (d:Devotee {Devotee ID: 'D001', Name: 'Raj kumar', Age: 30, phone: '9876543210', Email: 'rajkumar@gmail.com'}) RETURN d.

* CREATE a Temple Node

CREATE (t: Temple {Temple ID: 'T001', Name: 'Sri Venkateswar Temple', Location: 'Tirupati', Capacity: 5000}) RETURN t.

* Create a Booking Node

CREATE (b: Booking {Booking ID: 'B001', Devotee ID: 'D001', Temple ID: 'T00', Date: '2025-09-10', No. of Tickets: 2}) RETURN b.

2. Create Relationships:-

* Devotee Makes Booking

MATCH (d: ~~Book~~ Devotee {Devotee ID: 'D00'}), (b: Booking {Booking ID: 'B001'})

Output : Query.

Ticket ID	date	seat number	status.
TK1	2025-10-01	A10	Booked.

Find :-

TicketID	date	Seatnumber	status.
TK1.	2025-10-1	A10	Booked.

CREATE (a) – [: MADE] → (b).

RETURN d, b.

* Booking linked to Temple

MATCH (b: Booking {Booking ID: 'B001'}), (t: Temple {Temple ID: 'T001'})

CREATE (b) – [: FOR] → (t)

RETURN b, t

3. Display All Nodes.

MATCH (n) RETURN n.

4. Retrieve particular Booking.

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

5. Delete particular Booking

MATCH (d: Devotee {Devotee ID: 'D001'}) SET d. phone = '9123456780', d. Age = 31 RETURN d.

Update.

TicketID	date	seatnumber	status
TK1	2025-10-01	A10	Cancelled

Delete:

Ticket-ID	date	seatnumber	status.
(none)			

6. Delete Particular Booking

MATCH (b: Booking {Booking ID: '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	18.09.25

Result:- By following the above steps, was successfully Created, inserted, queries, updated and deleted of Temple Ticket Booking Management.

25/9/25

Task 10

Normalizing databases using functional depend--encies 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 Manage--ment 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, Time slot, Ticket type, Price, Payment ID, Payment mode, payment status, Transaction Date.
).

Step (a): Apply functional dependencies → Normalize to 1NF

1. User ID → U Name, U Email, U Contact
2. Temple ID → T Name, Temple Contact.
3. Priest ID → Priest Name, priest contact.
4. Ticket ID → Ticket ID, price.
5. Payment ID → Payment mode, payment status Transaction Date.

Step(d):-

Normalize to 2NF

Conditions for 2NF

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

Here, Booking ID is primary key so no partial dependency exists. the schema is in 2NF.

Step(e):- Normalize to 3NF

Conditions for 3NF

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

check:

* Booking ID → User ID → Uname.
* Booking ID → Temple ID → T Name.
* Booking ID → Priest ID → priest Name,
* Booking ID → Ticket ID → Ticket type, price.
* Booking ID → Payment ID → Payment mode, payment status.

Final simplified tables in 3NF

1. User table.

User (User ID, (PKJ, UName, U Email, Ucontact)

2. Temple table.

Temple (temple ID[PKJ, T Name, T location)

3. priest table.

priest (priest ID(PK), priest Name, priest constant),

4. ~~Ticket~~ Payment table.

Payment(Payment ID (PK), payment mode, Payment status, transaction date).

5. Ticket table.

Ticket (ticket ID (PK), ticket type, price).

Constraints :-

* primary key :

Booking ID, User ID, Temple ID, priest ID, Ticket ID, payment ID.

* Foreign keys :

→ Booking user ID → user user ID.

* Booking Temple ID → Temple . Temple ID.

* Booking . priest ID → priest . priest ID.

* Booking . Ticket ID → Ticket . ticket ID.

* Booking . Payment ID → Payment . Payment ID.

Unique constraints :-

* Price > 0.

* payment status ∈ ('pending', completed, 'failed').

* Time slot with in valid darshan timings.

VEL TECH - CSE	
EX NO	10
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	18
SIGN WITH DATE	

25/9

Result:- Thus the given relation for the online Booking management system has been normalized into simplified table upto 3NF.

9/10/25

Task-11

Design an Online Ticket booking system using Oracle Forcems menus, Report builder.

Aim:-

To design an online temple ticket Booking Management System using Oracle forms, menus and Report Builder.

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 form, define the data model that connects the database schema for the temple ticket booking system use.

* Temples
* Ticket types
* Booking details.
* User ID Devotees.
* Payment Information.

Set up data blocks for all neccessary data to manage ticket booking, user details and payments.

Design Forms:-

Forms are used to capture, display & Exit data Related to ticket booking.

Steps to design:-

1) Create a new form for each major component.
   * Ticket Booking.
   * User Registration
   * Payment processing
   * Booking cancellation form.

2) Add form Elements like text files, (for user Info), buttons(submit/cancel).

3) Use the property P/delete 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 tickets availability.
   * processing payments.

Create Reports:-

Report provide summarized information about booking and temple visits.

Steps to create:-

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.

reports:

* Daily ticket sales
* Booking summary by temple
* Revenue Report.

4) Design a report. layout with headers/ footer / and detailer data columns.
5) Add parameters to allow Retrieving data, range, temple, name.
6) Generate and preview the reports to verify the data actually and presentation.

VEL TECH - CSE	
EX NO	11
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	2
RECORD (5)	5
TOTAL (20)	17
SIGN WITH DATE	

9/10

Result:- The system is successfully delivered to faciliate online booking and efficient temple management.

Use Case 3: A gift coupon application that handles offers and payment-related information.

Choose the database system, a relational database, for its capability to scale horizontally while keeping the ACID property. The last was particularly important because transactional data was being handled. Eventual consistency as offered by other databases was not suitable. Going with DB and cluster gives the opportunity to scale horizontally for a large number of writes and reads without compromising the ACID and transactional capability required by the application. Will it transact and lead to no deadlock, keeping all relational tables normalized? If so what normal form do they sustain to offer a gift coupon application?

Team Members:

NEELISETTY VENKATA SAI THARUN

SUNKESWARA SUSHANTH

SHAIK GOUSE MOHIDDIN

DODLA SANTHOSH

MANNURU MEDHINI

KALLURI PEDDA BABU

CHANDRAJIT KAMALAKKANNAN

VADAMADHURA DILEEP KUMAR

YARASANI TARUN KUMAR REDDY

KOMMINA SRUTHI

GOSU BHARGAV RAM

SHAIK SHAHIL ROHAN

THONDAPU SAI JASWANTH

DASARI KOTISURYA

RAVURI NAGA DURGA PRASAD

BOARD OF SECONDARY EDUCATION ANDHRA PRADESH

DADIPINENI BHARGAVI

PANAGANTI SAI SRINATH

CHAMARTHI TEJESWAR RAJU

NAGANABOYINA MOHAN VENKAT

KANUPARTHI POOJITH REDDY

KONDAPATURI MADHU SUMANTH

KUCHANA SRAVIKA

VEPURALA VIPIN RAJ

GANAPATHIRAJU JAGANNADHA VARMA

KODURU KESAVA

GANDLURU VENKATA ARAVIND

ANISETTI ROHITHA

AIM:

Build a backend component that:

1. Stores coupons/offers and payment records.
2. Validates & applies coupons to orders (checks expiry, usage limits, min-order, allowed users).
3. Calculates final payable amount and records the payment + coupon usage.

ALGORITHM (high level)

1. Receive order request: (user_id, cart_amount, coupon_code, payment_method).
2. Load coupon by code. If not found → reject.
3. Validate coupon:
   - check active flag
   - check current_date ≤ expiry_date
   - check total_usage and per_user_usage limits
   - check min_order_amount
   - check if user is allowed (optional)
4. Compute discount:
   - if type = percentage → discount = min(cart_amount * pct/100, max_discount)
   - if type = fixed → discount = fixed_amount
5. Final_amount = max(cart_amount - discount, 0) + taxes/shipping (if any).
6. Process payment via gateway (simulate). If success:
   - create payment record (status = success)
   - increment coupon usage counters
   - return success + invoice

     Else:
   - create payment record (failed)
   - return failure reason

```
CREATE TABLE users (
 id INT PRIMARY KEY,
 name TEXT
);
```

```
CREATE TABLE coupons (
  id SERIAL PRIMARY KEY,
  code VARCHAR(50) UNIQUE,
  type VARCHAR(10),          -- 'percentage' or 'fixed'
  value NUMERIC,             -- percent or fixed amount
  max_discount NUMERIC NULL,
  min_order NUMERIC DEFAULT 0,
  expiry DATE,
  active BOOLEAN DEFAULT TRUE,
  usage_limit INT DEFAULT 0,   -- 0 = unlimited
  per_user_limit INT DEFAULT 1
);

CREATE TABLE coupon_usage (
  id SERIAL PRIMARY KEY,
  coupon_id INT REFERENCES coupons(id),
  user_id INT REFERENCES users(id),
  used_count INT DEFAULT 0
);

CREATE TABLE payments (
  id SERIAL PRIMARY KEY,
  user_id INT REFERENCES users(id),
  coupon_id INT NULL REFERENCES coupons(id),
  amount NUMERIC,
  discount NUMERIC,
  final_amount NUMERIC,
  status VARCHAR(20),  -- 'success'|'failed'
  method VARCHAR(50),
  created_at TIMESTAMP DEFAULT now()
```

1.);
   Find coupon:

SELECT * FROM coupons WHERE code = 'WELCOME10';

2. Check total usage:

SELECT COALESCE(SUM(used_count),0) AS total_used

FROM coupon_usage WHERE coupon_id = 123;

3. Get user's usage:

SELECT used_count FROM coupon_usage WHERE coupon_id = 123 AND user_id = 45;

4. Record successful payment:

INSERT INTO payments (user_id, coupon_id, amount, discount, final_amount, status, method)

VALUES (45, 123, 1000, 100, 900, 'success', 'card');

5. Increment user coupon usage (insert or update):

6. INSERT INTO coupon_usage (coupon_id, user_id, used_count)
7. VALUES (123,45,1)
8. ON CONFLICT (coupon_id, user_id) DO UPDATE
9. SET used_count = coupon_usage.used_count + 1;
10. **EXAMPLE: Concrete Coupon + Walkthrough**
11. Coupon row:

id	code	type	value	max_discount	min_order	expiry	usage_limit	per_user_limit
123	WELCOME10	percentage	10.00	150.00	500.00	2025-12-31	1000	1

12. User places order:

user_id = 45

- cart_amount = 1200.00
- coupon_code = WELCOME10

Validation:

- cart >= min_order (1200 >= 500) → OK
- percentage 10% → raw discount = 120.0 → below max_discount(150) → discount = 120.00
- final_amount before tax = 1200 - 120 = 1080.00

Assume tax 18% on final_amount (optional):

- tax = 1080 * 0.18 = 194.40
- payable = 1080 + 194.40 = 1274.40

Payment simulation success.

SAMPLE PROGRAM OUTPUT(Console-style)

Applying coupon: WELCOME10

User: 45 Cart: 1200.00

Coupon valid — percentage 10% (max 150.00)

Discount applied: 120.00

Amount after discount: 1080.00

Tax (18%): 194.40

Final payable amount: 1274.40

Payment status: SUCCESS

Payment ID: PAY_20251017_001

Coupon usage updated for user 45 (used_count = 1)

user_id = 45

- coupon_code = WELCOME10

Validation:

- cart >= min_order (1200 >= 500) → OK
- percentage 10% → raw discount = 120.0 → below max_discount(150) → discount = 120.00
- final_amount before tax = 1200 - 120 = 1080.00

Assume tax 18% on final_amount (optional):

- tax = 1080 * 0.18 = 194.40
- payable = 1080 + 194.40 = 1274.40

Payment simulation success.

SAMPLE PROGRAM OUTPUT(Console-style)

Applying coupon: WELCOME10

User: 45 Cart: 1200.00

Coupon valid — percentage 10% (max 150.00)

Discount applied: 120.00

Amount after discount: 1080.00

Tax (18%): 194.40

Final payable amount: 1274.40

Payment status: SUCCESS

Payment ID: PAY_20251017_001

Coupon usage updated for user 45 (used_count = 1)

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

16/10

Result:- Thus the build of backend components that stores coupons offers and payment records validates & applies coupons to order the calculates final payable amount and records completed successfully.