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

NAAC A++

Recognised as CATEGORY-1 DEEMED TO BE UNIVERSITY by the UGC University Grants Commission

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: THOTA BINDU

VTU.NO: 30569

REG.NO: 24UECL0058

BRANCH: CSE(AIML)

YEAR/SEM: II nd YEAR / III rd SEM.

SLOT: L12

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

NAAC A++

Recognized as CATEGORY-1 DEEMED TO BE UNIVERSITY by the UGC University Grants Commission

School of Computing
Department of Computer Science & Engineering
(Artificial Intelligence and Machine Learning)
ACADEMIC YEAR 2025-26 (SUMMER SEMESTER)

BONAFIDE CERTIFICATE

NAME : THOTA BINDU

VTU NO. : 30569

YEAR/SEM : II nd YEAR / III rd SEM

BRANCH : CSE(AIML)

REG.NO. : 24UECL0058

SLOT NO. : 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. Thyga Thi
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

No	Date	Title	Page No.	Marks	Faculty Signature
1.	24/07/25	Conceptual Design after Formal Technical Review	1	18	T.M.T 24.07.25
2.	31/07/25	Generating design of other traditional database models	5	18	T.M.T 31.07.25
3.	07/08/25	Developing queries with DML Single-row functions and operations	10	17	T.M.T 07.08.25
4.	14/08/25	Developing queries with DML Multi-row functions and operations	15	19	T.M.T 14.08.25
5.	21/08/25	Writing Join Queries, equivalent, and/or recursive queries	18	19	T.M.T 21.08.25
6.	28/08/25	Writing PL/SQL using Procedures, Function	22	19	T.M.T 28.08.25
7.	04/09/25	Writing PL/SQL using Loops	27	20	T.M.T 04.09.25
8.	11/09/25	Normalizing databases using functional dependencies up to BCNF	31	19	T.M.T 11.09.25
9.	18/09/25	Backing up and recovery in databases	34	18	T.M.T 18.09.25
10.	25/09/25	CRUD operations in Document databases	37	20	T.M.T 25.09.25
11.	09/10/25	CRUD operations in Graph databases	41	17	T.M.T 09.10.25
12.	16/10/25	Micro Project: use case 4: Army supply chain, Bill of Materials and Maintenance Cost Management.	45	18	T.M.T 16/10/25

Completed

Total Marks: 220/240

R.T. Tripathi

Signature of Faculty

DATE : 24-07-2025

Task-1

A Temple ticket online booking Management System

A temple ticket online booking Management system allows devotes 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 identical. examples included students courses, or products.

Entity Set:

A collection of entities of the same type. For instance, all students in a university would form an entity set

Attribute:

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 devotes to 10000 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, Data, 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, issue_Date, QR_Code, Amo Method, States, 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:

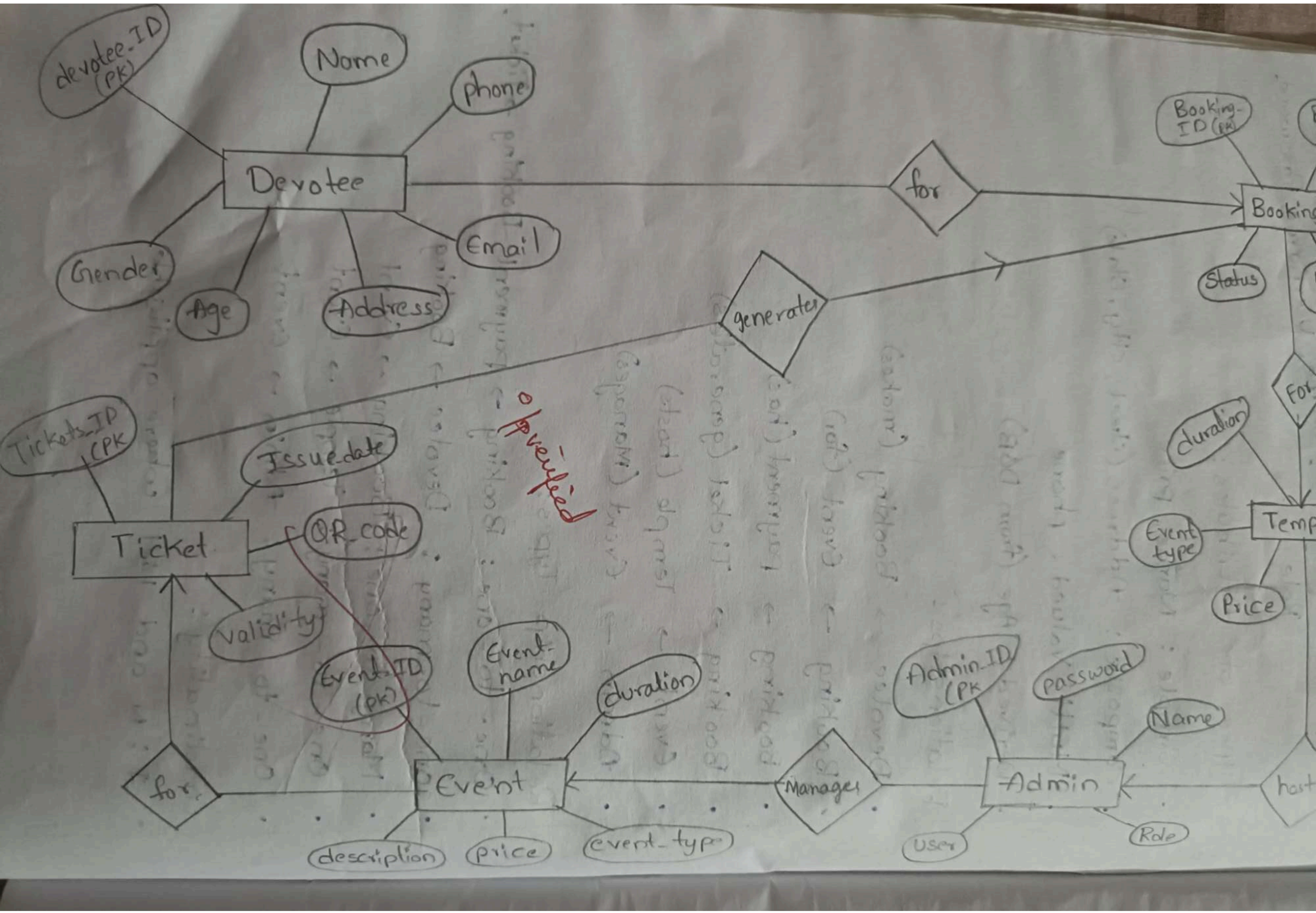
- Devotee → Booking (makes)
- Booking → Event (for)
- Booking → payment (has)
- Booking → Ticket (generates)
- Event → Temple (hosts)
- Admin → Event (Manages).

Relationship Types:

- One-to-one : Booking → payment, Booking → Ticke
- one-to-many : Devotee → Booking
- Many-to-one : Booking → Event
- One-to-many : Temple → Event
- One-to-many : Admin → Event.

Cardinality:

- 1 : N and 1 : 1 where applicable.


devotee_ID (PK)
Name
phone
Devotee
Gender
Age
Address
Email
for
Booking_ID (PK)
Booking
Status
generates
Tickets_ID (PK)
Issue_date
Ticket
QR_code
Validity
for
Event_ID (pk)
Event_name
duration
Event
description
price
event_type
Manages
Admin_ID (PK
password
Name
Admin
User
Role
duration
For
Event type
Price
host

VEL TECH - CSE

EX NO	1
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	3+5=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.

DATE : 31-07-25

Task 2 : Generating design of other traditional database model.

Aim : Creating Hierarchical/Network model of the data base by enhancing the sound abstract data by enhancing the sound abstract data by performing following tasks using forms of inheritance:

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

2.b. check is a Hierarchy / has-a - hierarchy and performs generalization and/or specialization relationship.

2c. Find the domain of the attribute and perform 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.

1. Relationship specificity:

a. one-to-many (Devotees ↔ Booking):

- Each devotee can have many bookings
- Each booking is uniquely linked to one devotee

b. (Booking ↔ ticket) one-to-one:

- one booking generates one or more tickets
- Ticket is dependent on Booking

c. One-to-one (Booking ↔ Event)

- Each booking is made for one event.

d. One-to-one or one-to many:

payment ↔ Booking:

- Each payment is tied to a booking

2. Surplus relations:

Here we identify entities that might be redundant, merged or consolidated for normalization.

a. Ticket may be merged with booking (1:1 or 1:N)

b. Event details redundant in booking

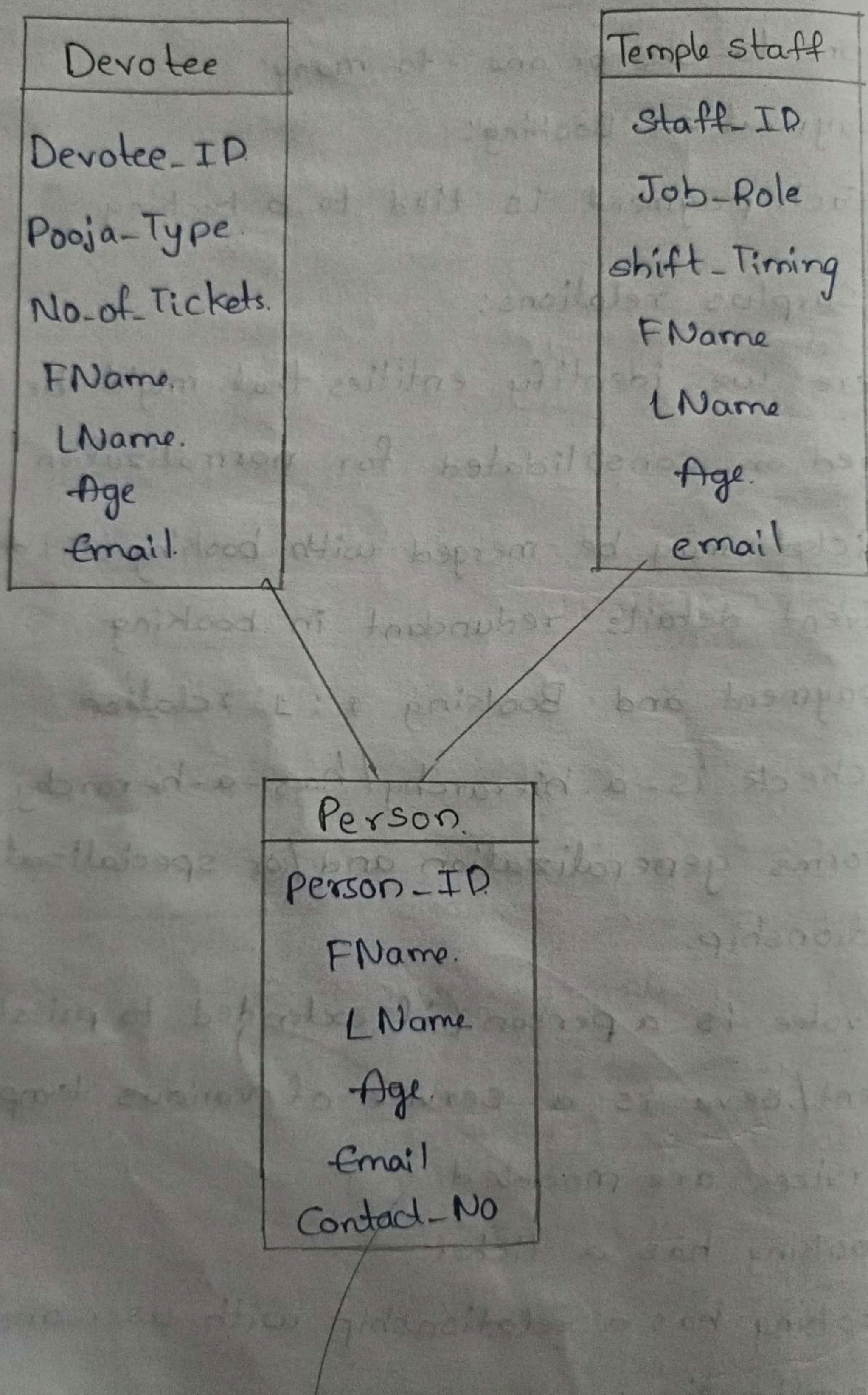
c. payment and Booking 1:1 relation.

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

- devotee is a person (if extended to priest, admin)
- Event/seva is a service of various temple.
- services are modeled.
- Booking has a ticket.
- Booking has a relationship with user and temple.

Generalization:

- person (general) → devotee (specialized)
- service → Event/seva.


Devotee
Devotee_ID
Pooja_Type
No_of_Tickets
FName
LName
Age
Email
Temple staff
Staff_ID
Job_Role
shift_Timing
FName
LName
Age
email
Person
Person_ID
FName
LName
Age
Email
Contact_No

Different types of users: Devotee user, Admin user.

2c. Attribute Domain & check constraints:

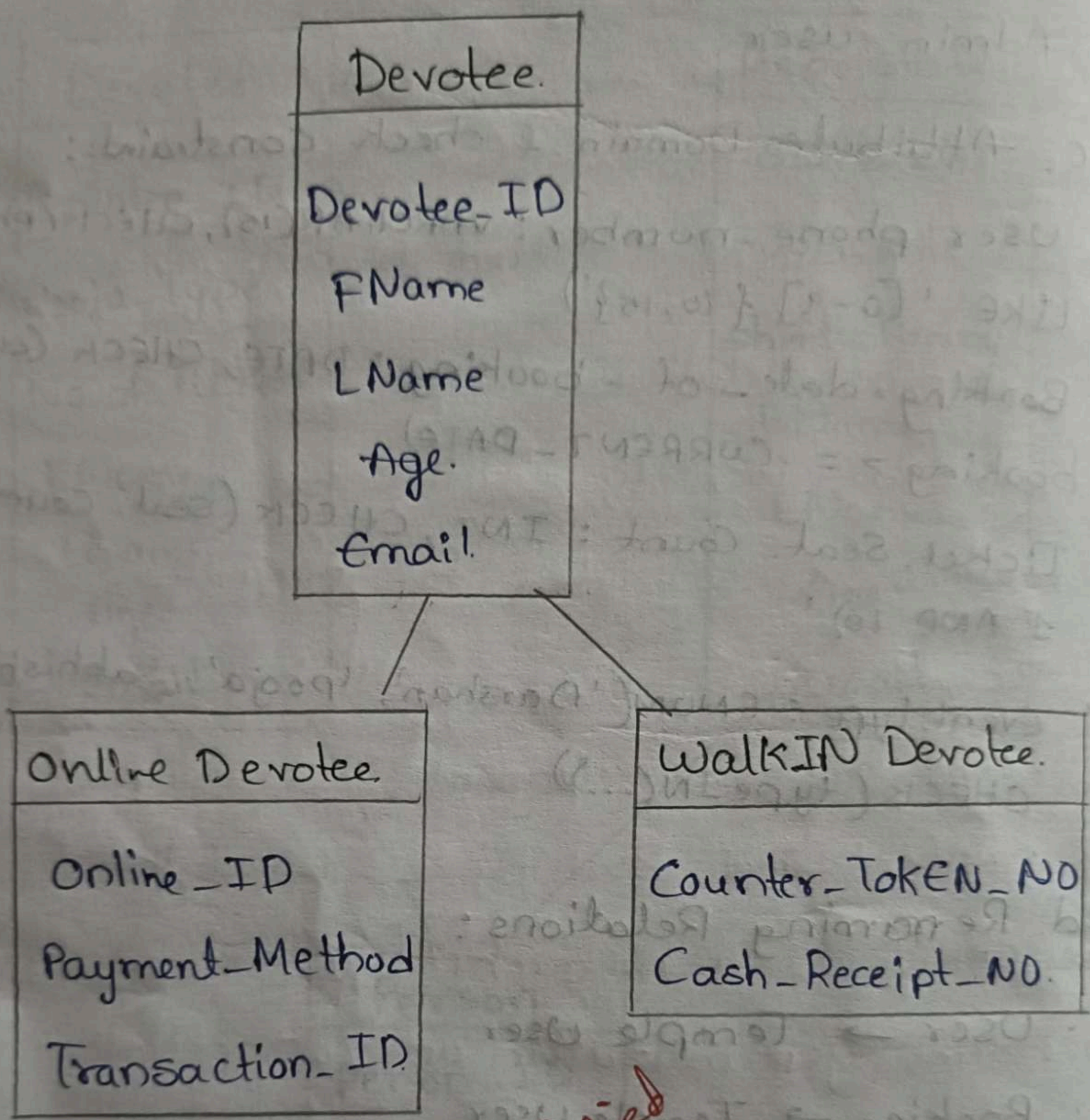
- User phone_number: VARCHAR(15), CHECK(phone_number LIKE '[0-9]{10,15}')
- Booking. date_of_booking: DATE, CHECK (date_of_booking >= CURRENT_DATE)
- Ticket. Seat_Count : INT, CHECK (seat_Count BETWEEN 1 AND 10)
- Event. type : ENUM('Darshan', 'pooja', 'abhishekam'), CHECK (type IN(...))

2d. Renaming Relations:

- User → Temple User.
- Booking → Temple User
- Ticket → Temple Ticket
- Event → Temple Event

2e. SQL Relations using DDL, DCL

```
CREATE TABLE Temple User(
        user_id INT PRIMARY KEY,
        name VARCHAR (100) NOT NULL,
        phone_number VARCHAR (15) NOT NULL
CHECK (phone_number ~'^[0-9]{10,15}$')
);
CREATE TABLE Temple(
```


Devotee.
Devotee_ID
FName
LName
Age.
Email.
Online Devotee.
Online_ID
Payment_Method
Transaction_ID
WalkIN Devotee.
Counter_TokEN_NO
Cash_Receipt_NO.

o/p verified

```
    temple_id INT PRIMARY KEY
    temple_name VARCHAR(100) NOT NULL,
    location VARCHAR(255)
);
CREATE TABLE TempleEvent(
    event_id INT PRIMARY KEY,
    temple_id INT REFERENCES
Temple (temple_id),
    event_type VARCHAR(20) CHECK
(event_type IN ('Darshan', 'pooja', 'Abhishekam')),
    event_data DATE,
    CONSTRAINT chk_event_date CHECK
(event_date >= CURRENT_DATE)
);
CREATE TABLE Temple Booking (
    booking_id INT PRIMARY KEY,
    user_id INT REFERENCES
Temple User (user_id),
    event_id INT REFERENCES
Temple Event (event_id),
    booking_date DATE DEFAULT
CURRENT_DATE
);
CREATE TABLE Temple Ticket(
    ticket_id INT PRIMARY KEY,
    booking_id INT REFERENCES
Temple Booking (booking_id),
```

```
seat_count INT CHECK (seat_count BETWEEN
                                    1 AND 10)
);
```

VEL TECH - CSE

EX NO	2
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	18+2=18
SIGN WITH DATE:	

31/07

Result:

Thus the Hierarchical model and network model has been successfully Created.

Date : 07-8-25

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, date functions, string functions, set clauses, and operators.

Temple.

TempleID	Name	location	Contact_No.
TID01	Tirumala Venkateswar	Tirupati	9876543210.
TID02	Meenakshi-Amman	Madurai	9867543210.
TID03	Kashi vishwanath	Varanasi	9856543210
TID04	Jagannath Temple	Puri	9846543210
TID05	Golden Temple	Amritsar	9836543210

Visitor:

visitor ID	FName	LName	Age	Email	Contact_No.
V001	Anil	Kumar	30	anilk@gmail.com	9123456780
V002	Akash	Reddy	25	akash@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:

BookingID	TempleID	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:

o/preferred

PriestID	FName	LName	Age	TempleID	Email	Contact
P001	Ramesh	Iyer	50	TID01	ramesh@gmail.com	9998887[illegible]
P002	Suresh	Sharma	45	TID02	suresh@gmail.com	9998887[illegible]
P003	Manish	Das	40	TID03	manish@gmail.com	9998887[illegible]

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

```
Select *
FROM visitor.
WHERE FName LIKE 'A%';
```

Result:

visitor ID	FName	LName	Age	Email	Contact_No.
V002	Akash	Reddy	25	akashr@gmail.com	9123456781
V004	Aarav	Patel	35	aaravp@gmail.com	9123456783

3. Add a column for 'Special_Seva' in Bookingtable.

```
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 details for TempleIDs 'TID01', 'TID03', and 'TID05'.

```
SELECT *
FROM Temple.
WHERE TempleID IN ('TID01', 'TID03', 'TID05');
```

Result:

TempleID	Name	Location	Contact_No.
TID01	Tirumala Venkateswara	Tirupati	9876543210
TID03	Kashi Vishwanath	Varanasi	9856543210.
TID05	Golden Temple.	Amritsar	9836543210.

6. Select VisitorID and names of visitors who booked 'Special' tickets.

```
SELECT VisitorID, FName, LName.
FROM Visitor.
WHERE VisitorID IN (
    SELECT VisitorID FROM Booking WHERE Ticket_Type =
                                        'Special'.
);
```

Result:

VisitorID	FName	LName.
V005	Sneha	Rao.

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

```
SELECT PriestID
FROM Priest
WHERE TempleID is NULL;
```

Result :

PriestID.

VEL TECH - CSE	
EX NO	3
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	4
TOTAL (20)	17
SIGN WITH DATE	

Result:

Thus, query processing for Temple Ticket Online Booking Management using Clauses, operators, and functions was successfully performed.

Date: 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 statistics in the Temple Ticket Online Booking Management System.

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

```
SELECT t. TempleID,
       t. Name As TempleName,
       t. Location,
       t. Contact_No,
       COUNT(b. BookingID) As Total Bookings
FROM Temple t
LEFT JOIN Booking b
ON t. TempleID = b.TempleID
Group By t. TempleID, t. Name, t. Location, t.Contact_No;
```

4.2: To retrieve the total number of 'special' seva bookings in a temple-wise manner.

```
SELECT t. Name. As TempleName,
       COUNT(*) As TotalSpecialBookings.
FROM Temple t.
```

Output:

TempleID	Temple Name	Location	Contact_No	Total Booking
TID01	Tirumala Venkateswara	Tirupati	9876543210	3
TID02	Meenakshi Amman	Madurai	9867543210	1
TID03	Kashi Vishwanath	Varanasi	9856543210	1
TID04	Jagannath Temple	Puri	9856543210	0
TID05	Golden Temple.	Amritsar	9836543210	0

```
JOIN Booking b.
ON t.TempleID = b.TempleID.
WHERE b.Ticket_Type = 'special'
GROUP BY t.Name;
```

4.3: To retrieve the details of temples where bookings include 'VIP' tickets.

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

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

```
SELECT v. visitorID,
        v.FName. As VisitorName,
        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:

TempleName	Total Special Bookings
Tirumala Venkateswara	1

Output:

TempleID	Name	location	Contact
TID01	Tirmula venkateswara	Tirupati	9876543210.
TID03	Kashi vishwanath	Varanasi	9856543210.

output:

visitorID	visitorName	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

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

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

```
SELECT t.TemplID,
       t.Name AS TemplName,
       v.FName AS visitorName
FROM Temple t
JOIN Booking b
ON t.TemplID = b.TemplID
WHERE visitor v
ON b.visitorID = v.visitorID
WHERE v.visitorID = 'V005';
```

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

17/10/25

Result:

Thus, the queries using functions, joins and nested subqueries were successfully executed for the Temple Ticket Online Booking Management System.

21/08/25.

Task-5 : Writing Join Queries, equivalent, and/ or recursive queries:

Aim:

To perform advanced query processing and test its heuristic 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 and outputs.

5.1 To retrieve all temples and their poojas.

Sql

```
SELECT t.Name AS Temple, p.PoojaName, p.Price
FROM Temple t.
JOIN Pooja p ON 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;
```

5.1 Output:

Temple	PoojaName	Price
Tirupati Temple	Suprabhata Seva	300
Tirupati Temple	Archana	100
Kanchi Temple.	Abhishekam.	500.
Madurai Temple	Special Darshan	250.

5.2 output:

BookingID	Devotee	PoojaName.	BookingDate	SlotTime	Status
B101	Rajesh.	Suprabhata Seva	22-JUN-2023	06:00AM	Confirmed
B102	Meena	Archana	22-JUN-2023	09:00AM	Confirmed
B103	Arjun.	Abhishekam	23-JUN-23	07:30AM	Cancelled
B104	Kavitha	Special Darshan	23-JUN-23	08:30AM	Confirmed
B105	Suresh	Abhishekam	24-JUN-23	06:30AM	Confirmed

5.3 output:

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

5.3 Count the number of bookings made by each devotee.

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

5.4 To find all devotees who booked 'Abhishekam'.

```
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: ~~The~~ To retrieve all poojas and the number of times they were booked.

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

5.6 To retrieve the total number of cancelled bookings temple-wise.

```
SELECT t.Name AS Temple, COUNT(b.BookingID)
AS CancelledBookings.
FROM Temple t.
```

5.4 Output

DevoteeID	FName	Mobile
D103	Arjun	9876543210
D105	Suresh	8765432190

5.5 output

PoojaName	TotalBookings
Suprabhata Seva	1
Archana	1
Abhishekam	2
Special Darshan	1

5.6 Output:

TempleID	Name	Location	HeadPriest
T001	Tirupati Temple	Tirupati	Ramanujam
T002	Kanchi Temple	Kanchipuram	Narayan Das
T003	Madurai Temple	Madurai	Subramanian

5.7 output

Temple	Cancelled bookings
Kanchi temple	1

5.8 Output:

FName	Age	BookingID	BookingDate	SlotTime	Status
Suresh	55	B105	24-JUN-23	06:30AM	Confirm

5.9 output:

TempleID	Name	location	Headpriest
T004	Rameshwaram Temple	Rameshwaram	Vishnu Devan.

5.10 output:

Temple	PoojaName	Devotee	BookingDate	SlotTime.
Tirupati Temple	Archana	Meena	22-JUN-23	09:00 AM.

```
JOIN Booking b ON t.TempleID = b.TempleID
WHERE b.Status = 'Cancelled'
GROUP BY t.Name;
```

5.7 To retrieve temple details where bookings were successful.

```
SELECT DISTINCT t.TempleID, t.Name, t.location,
t.HeadPriest.
FROM Temple t
JOIN Booking b ON t.TempleID = b.TempleID
WHERE b.Status = 'Confirmed';
```

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

```
SELECT d.FName, d.Age, b.BookingID, b.Booking
Date, b.SlotTime, b.Status.
FROM Devotee d
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

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

5.10 To retrieve temple, pooja, and devoote details for a given BookingID.

```
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 = '8102';
```

VEL TECH - CSE	
EX NO	5
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	4
TOTAL (20)	17
SIGN WITH DATE	

2/08

Result:

Thus, the Temple Online Booking Management System queries using Join Queries, equivalent queries, and recursive queries were executed succesfully to manage temples, devotees, priests, poojas, and bookings.

Date : 28-8-25

Task-6 : Procedures, Function and Loops. -
Temple online Booking.

Aim:

1. write PL/SQL block that calculates the average age of devotees and displays the result.
2. write a PL/SQL block that inserts a new booking into the booking Table.
3. Create a function that returns the Total number of bookings for a particular Temple.
4. Writeanon-recursive PL/SQL procedure to retrieve even-numbered Booking IDs registered for any Seva.

(1) PL/SQL block that calculates the average age of devotees and displays the result.

```
DECLARE
    v_total_age NUMBER:=0;
    v_count_devs NUMBER:=0;
    v_avg_age NUMBER:=0;
BEGIN
    -- Loop through all devotees and aggregate age.
```

1. Sample output.

Total Devotees : 120
Total Age : 3060
Average Age : 25.5.

8/8

2. Sample output.

Enter value for Booking ID : BKG203

Enter value for Devotee ID : DEV045.
Enter value for Temple ID : TMP001
Enter value for BDate : 2025-09-01
Enter value for slot : 06:00-07:00
Enter value for sevatype : Archana.
Enter value for Amount : 250.
Enter value for Status : CONFIRMED
Booking record inserted successfully.

3. output.

No. of bookings in temple : 42.

4. output.

Even_Numbered Booking ID : 102.
Even_Numbered Booking ID : 204
Even_Numbered Booking ID : 306

```
FOR rec IN(SELECT Age FROM Devotee) loop
  v_total_age := v_total_age + NVL(rec.Age,0);
  v_count_devs := v_count_devs + 1;
  END loop;
  If v_count_devs >0 THEN.
    v_avg_age := v_total_age/v_count_devs;
  END IF;
  -- Displaying the result
  DBMS_OUTPUT.PUT_LINE('Total Devotees:'|| v_count_
  devs);
  DBMS_OUTPUT.PUT_LINE('Average Age :'|| To_CHAR
  (ROUND(v_avg_age , 2)));
  END;
/
```

2) PL/SQL block that inserts a new booking into the Booking table.

```
DECLARE.
  V-BookingID VARCHAR2(10):='&BookingID';
-- e.g., 'BKG203'
  V-DevoteeID VARCHAR2(10):='&DevoteeID';
-- e.g., 'DEVO45'
  V-TempleID VARCHAR2(10):='&TempleID';
-- e.g., 'TMP001'
  V-BookingDate DATE :=
TO-DATE ('&BDate', 'YYYY-MM-DD');--
e.g., 2025-09-01
  V-slot VARCHAR2(20):='&slot';--
e.g., '06:00-07:00'
  V-sevatype VARCHAR2(30):='&sevatype';
--e.g; 'Archana'
  V-Amount NUMBER(10,2):=&Amount;
--e.g; 250.00.
  V_status VARCHAR2(15):='&status';
BEGIN.
  INSERT INTO Booking
  (BookingID, DevoteeID, TempleID, BookingDate, slot,
sevatype, Amount, status)
  VALUES.
  (V-BookingID, V-DevoteeID, V-TempleID, V-BookingDate, V-slot, V-sevaType, V-Amount, V-status);
  COMMIT;
```

```
DBMS_OUTPUT. PUT_LINE ('Booking record.
inserted successfully:');
EXCEPTION
   WHEN OTHERS THEN
   DBMS_OUTPUT. PUT_LINE ('Error:' || SQLERM);
   ROLLBACK;
     END;
   /
```

3) Function to return total number of bookings in a particular Temple.

```
CREATE OR REPLACE FUNCTION
Get Total Bookings IN Temple (P_TempleID
VARCHAR 2)
RETURN NUMBER.
IS.
 V_Total NUMBER; =0;
 BEGIN.
 SELECT COUNT (*) INTO V_Total.
   FROM Booking
   WHERE TempleID = P_TempleID;
   RETURN V_Total;
EXEEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN 0;
  WHEN OTHERS THEN.
    RETURN -1;
  END Get Total Bookings IN Temple;
  /.
```

4) Non-recursive procedure to retrieve even-numbered Booking IDs.

```
CREATE OR REPLACE PROCEDURE
Get Even Numbered Booking IDs.
IS.
BEGIN
  FOR rec IN (
    SELECT Booking ID
    FROM Booking.
    WHERE NOW (TO_NUMBER (Booking ID), 2) =0
    )
    loop.
    DBMS_OUTPUT.PUT.LINE ('Even-Numbered
Booking ID:' || rec. Booking ID);
  END loop;
  END Get Even Numbered Booking IDs;
/
Run the Procedure.
BEGIN.
  Get Even Numbered Booking IDs;
END;
/
```

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

20/8

Result:

Thus the PL/SQL Procedures, functions and loops on the Temple online Booking management system were successfully implemented and tested.

Date: 4/09/25

Task 7: Triggers, views and Exceptions.

Aim:

To ~~Coudt~~ conduct events, views, exceptions on CRUD operations for restricting phenomenon in a Temple online Booking Management system.

(a) To create a trigger in PL/SQL that automatically inserts a new record in the booking_audit table. when a new record is inserted into the bookings table.

(b) To create a view that displays the details of bookings along with the user and temple details.

(c) To write a non-recursive PL/SQL procedure to retrieve even-numbered BookingIDs registered for any Temple.

(a) Trigger

To create a trigger in PL/SQL that automatically inserts a new record in the booking_audit table. when a new record is inserted into the bookings table.

```
CREATE OR REPLACE TRIGGER
  insert_booking_audit.
  AFTER INSERT ON bookings.
    FOR EACH ROW
    BEGIN
```

```
INSERT INTO booking_audit (audit_id,
booking_id, user_id, temple_id, slot_id,
new_status, audit_ts)
        VALUES
(booking_audit_seq.NEXTVAL, :new.booking_id, :
new.user_id; :new.temple_id; :new.slot_id; :
NVL(:new.status, 'PENDING'), SYSDATE);
END;
/
```

(b) view

To create a view that displays the details of bookings along with the user and temple details.

```
CREATE OR REPLACE VIEW Booking Details As
SELECT b.booking_id,
       U.User_id,
       U.full_name. As User Name,
       U.mobile As User Mobile,
       t.temple_name,
       t.temple_id
       s.slot_id,
       s.slot_time,
       b.status,
       b.created_at.
FROM bookings b
JOIN users U ON b.user_id = U.user_id.
JOIN temples t ON b.temple_id = t.temple_id.
JOIN slots s ON b.slot_id = s.slot_id;
```

7.a) Execution ex:-

INSERT INTO Ticket Bookings
(username, Temple Name, pooja Name, Booking Date,
Booking Time, Number of people)
VALUES ('Ravi kumar', 'sri venkataswara
Temple', 'Abhishekam', TO_DATE ('2025-09-08',
'YYYY-MM-DD'), '10:00:00', 3);

O/P Ex:

Booking ID	User Name	Temple Name	Pooja Name	Booking Date
1.	Ravi Kumar	Sri venkateswara Temple.	Abhishekam	2025-09-25
2	Midhuna	Meenakshi Amman Temple	Madurai	2025-11-25.

O/P

To display:

```
SELECT * FROM Booking Details;
```

(c) Procedure.

To write a non-recursive PL/SQL procedure to retrieve even-numbered Booking IDs registered for any temple

```
CREATE OR REPLACE PROCEDURE
  Get Even Booking IDs for Temple(
  in_temple_id        IN NUMBER,
  out_Even_booking_ids OUT
  SYS.ODC IN NUMBER LIST.
  )AS
BEGIN
    out_Even_booking_ids :=
    sys.ODC IN NUMBER LIST();
FOR rec IN (SELECT booking_id
    FROM bookings
    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;
```

7.b)

SELECT * FROM Devotee Booking Details;

o/p:

Devotee ID Devoteename Email

Temple Name Ticket ID Booking Date visit Date

No. of tickets Amount

7.c)

Execution ex:-

BEGIN

GET Even Ticket ID D3 For Temple('TMP01');

END;

/

o/p:

Even Ticket ID : T102

Even Ticket ID : T104

Execution Example:

```
DECLARE
    temple_id NUMBER := 10;
    Even_ids SYS.ODC IN NUMBER LIST;
BEGIN
    Get Even Booking IDs For Temple (temple_id,
 even_ids);
    FOR i IN 1 .. Even_ids.COUNT loop
    DBMS_OUTPUT.PUT_LINE ('Even Booking ID:'
    || even_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)	15+5=20
SIGN WITH DATE	

Result:

Thus the Triggers, views and Exceptions experiment for Temple online Booking management system was successfully completed and results are verified.

Task-8: CRUD operations in Document Databases.

Aim: To perform CRUD using WPM design on MongoDB for Temple online booking management system, designing a document database and performing CRUD operations like creating, querying, finding, updating and deleting.

Steps:

Step 1: Install MongoDB using the following link:

https://www.mongodb.com/try/download/community.

Step 2: Install mongosh using the below link:

https://www.mongodb.com/docs/mongodb-shell/#download-and-install-mongosh

Step 3: To add mongoDB shell binary's location to your PATH Environment variable.

- 3.1 open the control panel.
- 3.2 Go to system and security → system
- 3.3 click Advanced system settings
- 3.4 click Environment variables.
- 3.5 In system variables, select path and
- ~~3.6~~ Click edit.
- 3.6 Add the filepath to your mongosh binary
- 3.7 click ok to confirm.

step 4 : open a command prompt and type mongosh -- help to confirm installation.

step 5 : open Mongo shell 4.0 from c:\ program files\mongoDB\server\bin\mongod.exe

step 6 : Type the CRUD (Create, Read, update Delete) commands in MongoDB.

CRUD operations

1. Create collection

db. Create collection ("Temple Booking")

{"ok": 1}

2. Insert one Record.

db. Temple Booking. insert one (

{ Booking ID: "B001", Templename: "Sri Venkateswara Temple", location: "Tirupati", Devotee Name: "Ramesh", phone : 9876543210, Booking Date: "2025-09-10"}.

)

3. Insert many records.

db. Temple Booking. insert Many ([

{ Booking ID: "B002", Temple Name: "Meenakshi Amman Temple", location: "Madurai", Devotee Name: "Suresh", phone: 9876501234, Booking Date: "2025-09-12"},

{ Booking ID: "B003", Temple Name: "Kapaleeshwarar Temple", location: "chennai", Devotee Name: "Priya", phone: 9966332211, Booking Date: "2025-09-15"},

2. output.

{"acknowledged": True, "insertedId": ObjectId("651CIF726befc7993adf901")}

3. output:

{
"acknowledged": True,
"insertedIds": [
objectid("651CF726befe7993adf902"),
objectid("651CF726bfe7993adf903"),
objectid("651CIF726bebfe7993adf904")
]

4. output.

{"_id": objectid("651cif726bcbfe7993adf901"), "Booking ID": "Boo1", "Temple Name": "Sri venkateswara Temple", "location": "Tirupati", "Devotee Name": "Ramesh", "phone": 9876543210, "Booking Date": "2025-09-10"}

{"_id": objectid("651cif726befe7993adf902") "Booking ID": "Boo2", "TempleName": "Meenakshi Amman Temple", "location": "Madurai", "Devotee Name": "Suresh", "phone": 9876501234, "Booking Date": "2025-09-12"}

{ Booking ID : "B004", Temple Name : "Anna malaiyar Temple", location: "Thiruvannamalai",
Devotee Name : "Anitha", Phone : 9955886644,
Booking Date : "2025-09-20"}
])

4. find records

db. TempleBooking. find()

5. Pretty format find.

db. TempleBooking. find(). pretty()

6. update a record.

db. TempleBooking. update one(
{ Booking ID : "B003"},
{ $set : { phone : 9123456789,
Devotee Name : "priya sharma"}}
)

7. Delete a record.

db. TempleBooking. delete one({ Booking ID : "B004"})

EXPERIMENT PROCEDURE	8
PROGRAM	5
RESULT AND ANALYSIS (5)	5
VIVA	4
RECORD (5)	5
TOTAL (20)	14+5=19
SIGN WITH DATE	G.M.Thr 11.09.25

Result: Thus, CRUD operations using NPM design of MongoDB for Temple online Booking Management system were successfully implemented, including creating, inserting, querying, updating, and deleting records.

5. output:

{"_id": objectId ("651c1f72bebfe7993adf903"),
"Booking ID": "B003",
"Temple Name": "Kapaleeshwarar Temple",
"location": "chennai",
"DevoteeName": "Priya",
"phone": 9966332211,
"Booking Date": "2025-09-15"
}

6. output

{"acknowledged": True, "matched count": 1,
"modified count": 1}

O/P

7. output

{"acknowledged": True, "deletedCount": 1}

DATE: 18/09/25.

Task-9 : CRUD operations in Graph databases.

Aim :

To perform CRUD operations like creating, quering, updating, and deleting on graph spaces for an online temple ticket Booking management system.

Step 1 : set up Neouj-AuraDB.

* click start free → continue with google → open the file
* copy the downloaded password into and login to the Aura console.
* start executing queries in the Neouj Browser.

Create nodes with properties.

each node represents an entity like devotee, Temple and Booking.

Create a Devotee Node :

CREATE (d: Devotee {Devotee ID : 'D007',
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.

→ Create a Booking Node

CREATE (b: Booking { BookingID: 'B001',

DevoTee ID : 'D007',

Temple ID : 'T007',

Return b.

Create Relationships

→ Devotee makes Booking.

MATCH (d: Devotee { DevoteeID : 'D001'})

CREATE (d) - [: MADE] → (b)

RETURN d, b.

→ Booking linked to Temple.

MATCH (b: Booking {BookingID: 'B007'})

CREATE (b) - [: FOR] -> (t)

RETURN b, t.

Display All Nodes

MATCH (n) RETURN n.

Retrieve particular Booking Details.

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

RETURN b.

Update particular Devotee Details.

MATCH (d: Devotee {DevoteeID : 'D007'})

SET d.phone = '9876543210', d.Age = 31

RETURN d.

Output

d.devoteeID	d.Name	d.Age	d.Phone
D001	RajKumar	30	9876543210

output

t.TempleID	t.Name	t.location	t.Capacity
T007	Srivenkatesh -wara Temple	Tirupati	5000

DELETE Particular Booking.

```
MATCH (b; Booking { BookingID: 'B001' })
DELETE b
```

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

18/9

Result:

By following the above steps we successfully created, inserted, queried, updated, and deleted nodes and relationships in the temple ticket Booking Management system using Neouj Graph Database

DATE: 25/09/25.

Task-10: Normalizing databases using functional dependencies up to Third Normal.

Aim:

To normalizing the below relation and create the simplified tables with suitable constraints for the online Temple ticket Booking management system.

Given Relation:

Temple Booking (
BookingID, User ID, UName, UEMail, UContact, TempleID, TName, Tlocation, PriestID, priestName, priest Contact, ticket ID, Booking Date, Darshan Date, Timeslot, Ticket type, price, payement ID, payment Mode, Payment status, Transaction Date.
)

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

first, identify functional dependencies (FDs):

1) User ID → UName, UEmail, UContact.
2) Temple ID → T. Name, Tlocation.
3) Priest ID → Priest Name, priestcontact.
4) Ticket ID → Tickettype, price
5) Payment ID → Payment mode, payment status

step (b): Normalize using FD+ and α+

Candidate Keys:

BookingID (Main unique identifier of a booking).

from $\alpha+$ (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 → UName, UEmail, Ucontact.
2. Temple ID → TName, Tlocation.
3. Priest ID → PriestName, Priest contact.
4. Ticket ID → Ticket type, price.
5. Payment ID → Payment Node, payment status.

step (d): Normalize to 2NF.

Conditions for 2NF:

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

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

Step (e): Normalize to 3NF.

Conditions for 3NF:

- Must already be in 2NF.
- No transitive dependencies.
- Booking ID → User ID → UName.
- Booking ID → Temple ID → TName.
- Booking ID → Priest ID → Priest Name.

- Booking ID → Ticket ID → Ticket Type, price.
- Booking ID → Payment ID → Payment mode

Final Simplified Table in 3NF.

1) User Table.

User (UserID [PK], UName, U Email, U contact)

2) Temple Table.

Temple (TempleID [PK], TName, Tlocation)

3) Priest Table.

Priest (priest ID [PK], priest Name, priestcontact)

4) Ticket Table.

Ticket (Ticket ID [PK], tickettype, price)

5) Payment Table.

~~Payment~~ (Payment ID [PK], Payment mode, Payment states)

6) Booking Table.

Booking (Booking ID [PK], user ID [FK],)

Constraints:

- Primary keys: Booking ID, ~~Ex~~ User ID, Temple ID, Priest 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 :- U Email, U Contact.
- 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	

25/9

Results:

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 redundancy and improving data integrity.

DATE: 09/10/25.

Task-11 : Menus, 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 booking tickets online, managing bookings, and generating reports related to ticket sales and temple visits.

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 book 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 from 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 (e.g., Book Tickets → Select Temple → Choose Ticket Type).
5. Assign triggers or procedures to handle menu item actions (e.g., open booking form).
6. Compile and run the menu form 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 bussiness logic, such as:
   - Validating booking dates.
   - checking ticket availability
   - Processing payments
5. Test the forms inside Oracle Forms Builder for functionality and data accuracy.

Create Reports:

Reports provide summarised information about bookings and temple vists.

Steps to create reports using Oracle Report Builder:

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. Examples reports:
   - Dialy Ticket Sales Report
   - Booking Summary by Temple
   - Revenue Report.
4. Design the Report layout with headers, footers, and detailed data columns (e.g., booking date, ticket type, amount).
5. Add parameters to allow filtering (e.g., data range, temple name).
6. Generate and ~~prie~~ preview the report to verify the data accuracy and presentation.

Output:-

Devotee table:-

Devotee_ID	Name	Mobile_NO	Email
101	RAM kumar	9976328310	Ram@gmail.com
102	sitadevi	9876501234	sita@gmail.com

Booking table:-

Booking_ID	Devotee_ID	POOJA_NAME.
201	101	Ganapathi thanam
202	102	lakshmi POOJA.

0/2

VEL TECH - CSE	
EX NO	11
PERFORMANCE (5)	4
RESULT AND ANALYSIS (5)	4
VIVA VOCE (5)	4
RECORD (5)	4
TOTAL (20)	12
SIGN WITH DATE	15-09-2025

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.

Use Case-04 DATE : 16/10/25

Army Supply Chain, Bill of Materials and maintenance Cost Management.

Team Members :

1. P. Harika
2. M. Rama Tulasi
3. T. Bindu.
4. G. Yasaswini
5. D. Pranathi
6. G. vijaya Lakshmi
7. P. Satvik
8. M. Bhanu Teja
9. J. Mahesh.
10. K. Soomanath Reddi
11. A. Bhanuprasad Reddy
12. A. Kesava Venkata Durga Ajay
13. B. Purna Ajay
14. Y. shyam Dhanush
15. T. Bhanu prasad
16. R. Yogesh Kumar.

Aim:

To design a data-driven model that efficiently manages and analyses the Army's supply chain, Bill of materials (BOM), and equipment maintenance costs using advanced data management and analytics techniques.

Description:

The nation's armed forces consist of over one million soldiers and approximately 2,00,00 civilian staff. Each personal depends on various equipment such as helicopters, armored vehicles, small arms, communication devices, and other critical resources. Since maintenance, operation, and support costs account for nearly 80% of the total lifecycle costs, it is crucial for the Defence Ministry to accurately track, analyze, and forecast maintenance and supply requirements.

Objectives:

1) To integrate and manage Army equipment data, maintenance logs, and BOM efficiently.
2) To forecast spare part requirements based on environmental and operational conditions.
3) To evaluate the mean time to failure equipment.

4). To perform multi-dimensional cost comparison and trend analysis.

5) To provide decision-makers with "what-if" scenario simulations for deployment.

Actors:

- Defence Data Analysts
- logistics offices.
- Maintenance engineer.
- Command Head quarters,
- Supply chain Manager.

Preconditions:-

- All equipment, maintenance, and logistic data are digitized and stored in a central system.
- The system must support integration with graph databases and analytics tools.

Post Conditions:

- optimized resource allocation and maintenance scheduling
- Reduced unpredictable maintenance costs through predictive analytics.
- Improved forecasting accuracy for spare parts and supply chain managements.

Output:-

- Graphical dashboards showing cost trends and failure rates.
- Forecasted spare parts and maintenance schedules
- Comparative reports on deployment and maintenance costs under different scenarios.

Use case Flow:

Step	Description.
1.	The Defence Data Analyst i/p and consolidates equipment maintenance data
2	The system integrates data from historical and real-time sources
3.	Predictive analytics identifies potential equipment failures and spare past needs.
4.	The system performs "what-if" analytics for deployment and logistics cost simulation.
5.	Reports and dashboards are generated for decision-makers to plan operators.

VEL TECH - CSE	
EX NO	06
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 Army supply chain and Maintenance cost MS provides a robust analytical solution.