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: M. Rama Tulasi

VTU.NO: VTU27599

REG.NO: 24UECL0031

BRANCH: CSE(AI&ML)

YEAR/SEM: S S2526-3

SLOT:

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 :	M. Rama Tulasi	**BRANCH** :	CSE (AI & ML)
VTU NO. :	VTU27599	**REG.NO.** :	24UECL0037
YEAR/SEM :	SS2526-3	**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).

3/10/25

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

lo	Date	Title	Page No.	Marks	Faculty Signature
1.	24/07/25	Conceptual Design after Formal Technical Review	1	17 17	T.M.T 24.07.25
2.	31/07/25	Generating design of other traditional database models	7	18 18	T.M.T 31.07.25
3.	07/08/25	Developing queries with DML Single-row functions and operations	11	18	T.M.T 0208.25
4.	14/08/25	Developing queries with DML Multi-row functions and operations	15	17	T.M.T 14.08.25
5.	21/08/25	Writing Join Queries, equivalent, and/or recursive queries	19	17	T.M.T 21.08.25
6.	28/08/25	Writing PL/SQL using Procedures, Function	23	20	T.M.T 28.08.25
7.	04/09/25	Writing PL/SQL using Loops	27	18 18	T.M.T 04.09.25
8.	11/09/25	Normalizing databases using functional dependencies up to BCNF	30	18	T.M.T 11.09.25
9.	18/09/25	Backing up and recovery in databases	33	17	T.M.T 18.09.25
10.	25/09/25	CRUD operations in Document databases	36	19	T.M.T 25.09.25
11.	09/10/25	CRUD operations in Graph databases	41	20	T.M.T 09.10.25
12.	16/10/25	Micro Project: Usecase 4:- Army Supply Chain, Bill of Materials and Mainterance Cost Management.	45	18	T.M.T 16/10

Total Marks: 217 / 240

T.M.T

Signature of Faculty

Task - 01 Date: 01/09/25

Conceptual Design After ERD

Temple Ticket Booking Management System

-Aim: Using basic database design methodology and ER modeling, design an Entity Relationship Diagram (ERD) by completing the following task.

1.a Identifying the Entities

1. Temple
2. Devotee
3. Pooja/Event
4. Ticket
5. Booking
6. Payment

1.b Identifying the Attributes

1. Temple (Temple ID (PK), Name, Location, DeityName, Contact_No).
2. Devotee (Devotee ID (PK), FName, LName, Gender, Age, Email, Phone)
3. Pooja (PoojaID (PK), TempleID (FK), Name, Date, Time, Price, Duration).
4. Ticket (TicketID (PK), PoojaID (FK), TicketType, Price, Availability).
5. Booking (Booking ID (PK), DevoteeID (FK), TicketID (FK), BookingDate, Quantity).
6. Payment (PaymentID (PK), BookingID (FK), Amount, PaymentMode, PaymentDate, Status).

1.c Relationships, Cardinality, and Type

Relationship	Entities Involved	Type	Cardinality
Temple-Pooja	Temple to Pooja	1:M	one temple - many poojas
Pooja-Ticket	Pooja to Ticket	1:M	one pooja - many ticket types
Devotee-booking	Devotee to Booking	1:M	one devotee - many bookings
Booking-Ticket	Booking to Ticket	M:1	each booking refers to one ticket
Booking-Payment	Booking to Payment	1:1	each booking has one payment

1.d SQL Table Definitions for Entities and Relationships

1.d.1 Temple Table

```
CREATE TABLE Temple (
    TempleID VARCHAR(10) PRIMARY KEY,
    Name VARCHAR(50),
    Location VARCHAR(100),
    DeityName VARCHAR(50),
    Contact_No VARCHAR(15)
);
```

Column Name	Null	DataType
TempleID	No	VARCHAR(10)
Name	Yes	VARCHAR(50)
Location	Yes	VARCHAR(100)
DeityName	Yes	VARCHAR(50)
Contact_No	Yes	VARCHAR(15)

1.d.2 Devotee Table

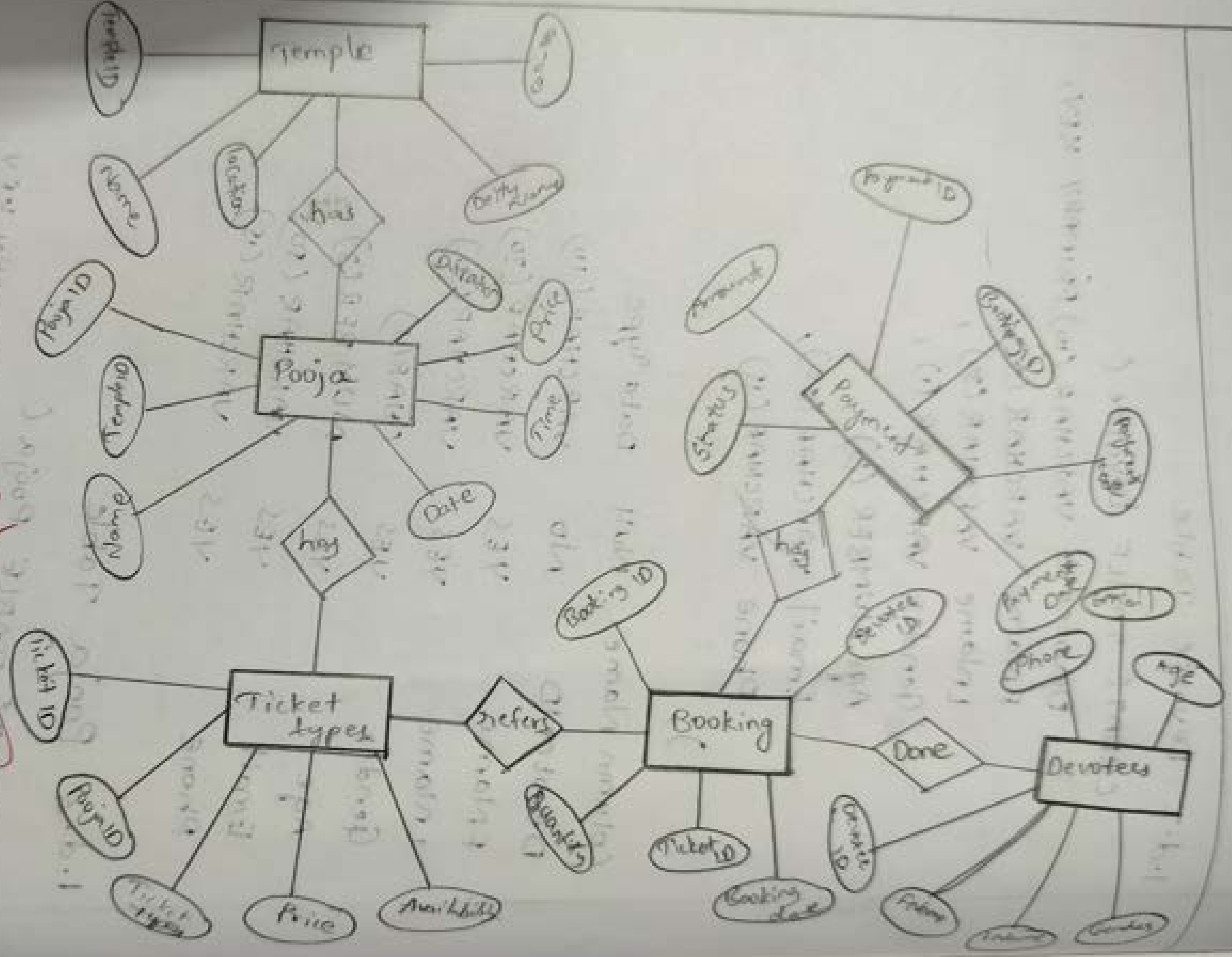
```
CREATE TABLE Devotee (
    DevoteeID VARCHAR(10) PRIMARY KEY,
    FName VARCHAR(30),
    LName VARCHAR(30),
    Gender VARCHAR(1),
    Age NUMBER(3),
    Email VARCHAR(50),
    Phone VARCHAR(15)
);
```

Column Name	Null	Data Type
DevoteeID	NO	VARCHAR(10)
FName	YES	VARCHAR(30)
LName	YES	VARCHAR(30)
Gender	YES	CHAR(1)
Age	YES	NUMBER(3)
Email	YES	VARCHAR(50)
Phone	YES	VARCHAR(15)

1.d.3 Pooja Table

```
CREATE TABLE Pooja (
    PoojaID VARCHAR(10) PRIMARY KEY,
    TempleID VARCHAR(10),
    Name VARCHAR(50),
    Date DATE,
    Time TIME,
    Price NUMBER(8,2),
    Duration VARCHAR(20),
    FOREIGN KEY (TempleID) REFERENCES
Temple(TempleID)
);
```

ER Diagram ⇒



Temple
TempleID
Name
location
has
Deity Name
Pooja
PoojaID
TempleID
Name
Duration
Price
Time
Date
has
Ticket types
Ticket ID
PoojaID
Ticket types
Price
Availability
refers
Booking
Booking ID
Devotee ID
Quantity
Ticket ID
Booking date
has
Payment
Payment ID
Amount
Status
Booking ID
Payment mode
Payment Date
Done
Devotees
Devotee ID
Email
Phone
Age
Gender
verified

Column Name	Null	DataType
PoojaID	NO	VARCHAR(10)
TempleID	YES	VARCHAR(10)
Name	YES	VARCHAR(50)
Date	YES	DATE
Time	YES	TIME
Price	YES	NUMBER(8,2)
Duration	YES	VARCHAR(20)

1.d.4 Ticket Table

```
CREATE TABLE Ticket (
    TicketID VARCHAR(10) PRIMARY KEY,
    PoojaID VARCHAR(10),
    TicketType VARCHAR(20),
    Price NUMBER(8,2),
    Availability NUMBER,
    FOREIGN KEY (PoojaID) REFERENCES
    Pooja (PoojaID)
    );
```

Column Name	Null	DataType
TicketID	No	VARCHAR(10)
PoojaID	YES	VARCHAR(10)
TicketType	YES	VARCHAR(20)
Price	YES	NUMBER(8,2)
Availability	YES	NUMBER

1.d.5 Booking Table

```
CREATE TABLE Booking (
    BookingID VARCHAR(10),
    DevoteeID VARCHAR(10),
    TicketID VARCHAR(10),
    BookingDate DATE,
    Quantity NUMBER,
    FOREIGN KEY (DevoteeID) RE
Devotee (DevoteeID),
    FOREIGN KEY (TicketID)
Ticket (TicketID)
);
```

Column Name	Null	Data Type
BookingID	No	VARCHAR (10)
DevoteeID	YES	VARCHAR (10)
TicketID	YES	VARCHAR(10)
BookingDate	YES	DATE
Quantity	YES	NUMBER

1.d.6 Payment Table

```
CREATE TABLE Payment (
    PaymentID VARCHAR(10) PRIMARY KEY
    BookingID VARCHAR(10),
    Amount NUMBER(10,2),
    PaymentMode VARCHAR(20),
    PaymentDate DATE,
    Status VARCHAR(15),
    FOREIGN KEY (BookingID) REFERENCES
Booking(BookingID) REFERENCES
);
```

Column Name	Null	DataType
PaymentID	NO	VARCHAR(10)
BookingID	YES	VARCHAR(10)
Amount	YES	NUMBER(10,2)
PaymentMode	YES	VARCHAR(20)
PaymentDate	YES	DATE
Status	YES	VARCHAR(15)

VEL TECH - CSE	
EX NO	1
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	2
RECORD (5)	5
TOTAL (20)	17
SIGN WITH DATE	

24/7/25

Result:- Thus, the database design methodology and ER model diagram for the online Temple Ticket Booking Management System has been completed successfully with all entities, relationships, constraints, and SQL schema creation.

Task No:- 02 Date:- 31/7/25

Generating Design of other traditional database model.

Aim:- Creating Hierarchid/ network model of the database by enhancing the sound abstract data by performing following tasks using forms of intesitance.

2.a) Identify each relationship find and form surplus relations.

Relationship	Type	Surplus relation
Temple conducts pooja	1:M (one to many)	Not required
Devotee books ticket	M:1 (many to one)	Not required
Ticket belongs to pooja	M:1 (many to one)	Not required
Booking has payment	1:1 (one to one	Not required
Admin manages temple	1:M (one to many)	Not required.

2.b) Check is-a hierarchy / has a hierarchy and perform generalization / or specialization relationship.

Entities:
Devotee
Admin
Common attributes
Name
Email
Con-No.
Generalized Entity
Person (Person ID, Name, Email, Contact-No).

Subclasses:-
Devotee :- Inherits from person and adds attributes (devotee, gender, Age, Address)
Admin :- Inherits from person and adds attributes like AdminID, password.

Entity:
Pooja
Specialized Subtypes:
. Archana Pooja : Addition attribuds like flowers, Mantras, Duration
. Abhishekam pooja: Attributes Lyk Water, milk etc.
This specialization provides flexibility to represent diff pooja types with their own specific attributes while still maintaining general pooja details in the parent entity.

2.c) Find the domain of the attribute and perform check constraint to the applicable

Let's take the age attribute in the Devotee as an example.

Domain of age: The valid age for booking should blw 10-100 yrs.

Constraint applied:- Check constraint is applied to ensure age is entered blw 10 - 100.

Action taken:- A constraint is applied to ensure the value of age is below 10 or above 100. if invalid age is entered, system will reject.

2.d) Rename the relations.

Renaming relations and columns can help improve clarity and standize naming

Example:-

Old column name: Contact_No in Devotee

New column name: Phone_No

Action taken
Column renamed successfully

Updated table structure

Name	Null	Type
DevoteeID	Not Null	varchar2(10)
FName		varchar2(30)
LName		varchar2(30)
Age		Number(3)
Gender		varchar2(10)
Address		varchar2(50)
Email		varchar2(40)
Phone_No		Number

2.e) Perform SQL Relations using DDL, DCL commands DCL (Data control language).

DCL is used to manage privileges on database objects.

Step 1 :- A new user named Meena was created to access the system.

Step 2 :- Privileges were granted to meena to create sessions and resources

Step 3 :- Meena logged in & created a table for storing temple feedback

Step 4 :- The system user granted privilege the temple table to Meena.

Commands performed:

- User created
- Resource role granted
- Create session granted
- Table created
- All privileges granted on Temple table.

Results All DCL operations completed successfully

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

Result: Thus the Hierarchical Model and Network model for online temple Ticket Booking Management System has been successfully completed.

Task No:- 03 Date:- 7/8/25

7 Using Clauses, Operators and Functions in queries

Aim:- To perform the query processing on databases for different retrieval results of queries using DML, DRL operations using aggregate, date, string, indent functions, set clauses and operators.

Tables in online Temple Ticket Booking Management System

Temple

TempleID	Name	Location	DeityName	Con-No
T001	Meenakshi Temple	Madurai	Meenakshi	9876543210
T002	Brihadeshwarar Temple	Thanjavur	Shiva	9898989898
T003	Tirupati temple	Tirupati	Venkateswara	9765432109
T004	Kashi Viswanath	Varanasi	Shiva	9123456780
T005	Srirangam Temple	Trichy	Ranganatha	9345678901

Booking

BookingID	TempleID	VisitorName	Visitor Email	Booking Date	Ticket count
B001	T1D01	Anitha	anitha@gmail.com	2025-8-1	2
B002	T1D03	Arjun	arjun@gmail.com	2025-08-5	4
B003	T1D02	Ramesh	ramesh@gmail.com	2025-8-3	1
B004	T1D04	Akash	akash@gmail.com	2025-8-2	3
B005	T1D05	Meera	meera@gmail.com	2025-8-4	2

Priest

PriestID	TempleID	Fname	Lname	Age	DOB	Rob	Speciality
P001	T1001	Kumar	S	45	1980-6-12	H.P	Abhishekam
P002	T1003	Mani	K	40	1985-4-10	Priest	Alankaram
P003	T1002	Rajesh	M	50	1975-2-20	Head Priest	Vedic chant
P004	T1004	Aranda	R	38	1987-9-15	Priest	Archana
P005	T1005	Viral	S	42	1983-11-25	Priest	Deepa Aradhana

Queries & Results

3.1) To retrieve the location of a particular booking's temple by its BookingID

Sql

```
SELEST Location
FROM Temple
WHERE TempleID=(SELECT TempleID FROM Booking
WHERE BookingID = 'B003');
```

Result;

Location
Thanjavur

3.2) To retrieve the bookings where VisitorName starts with 'A'.

Sql

```
SELECT *
FROM Booking
WHERE VisitorName LIKE 'A%';
```

Result:

Booking ID	Temple ID	Visitor Name	Visitor Email	Booking Date	Ticket count	Status
B001	T1D01	Anitha	anitha@gmail.com	25-8-1	2	confiamed
B002	T1D03	Arjun	arjun@gmail.com	25-8-5	4	confiamed
B004	T1D04	Akath	akath@gmail.com	25-8-2	3	confiamed

3.3) Add a column Speciality in Pricst table.

Sql

```
ALTER TABLE Pricst ADD Speciality VARCHAR(50);
```

Result:
Table Altered

3.4) To count the number of confirmed bookings

Sql

```
SELECT COUNT(*)
FROM Booking
WHERE status = 'confiamed';
```

Result:

```
count(*)
3
```

3.5) To display the Temple details for TempleIDs 'T1D01', 'T1D03', and 'T1D05'.

Sql

```
SELECT *
FROM Temple
WHERE TempleID IN ('T1D01', 'T1D03', 'T1D05');
```

Result:

TempleID	Name	Location	Deity Name	Con. No
TID01	Meenakshi Temple	Madurai	Meenakshi	9876543210
TID03	Tirupathi Temple	Tirupati	Venkateswara	9765432109
TID05	Srirangam Temple	Trichy	Ranganatha	9345678901

3.6) To select the names and IDs of priests who have speciality in 'Archana'.

Sql

```
SELECT PriestID, FName, LName
FROM Priest
WHERE Speciality = 'Archana';
```

Result:

PriestID	FName	LName
P004	Ananda	R

VEL TECH - CSE	
EX NO	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	4.5
TOTAL (20)	18
SIGN WITH DATE	

Result: Thus, the query processing on database for different retrieval results of queries using Clauses, Operators, and Functions in Online Temple Ticket BMS has been performed successfully

Task No:- 04 Date:- 14/8/25

Using Functions in Queries and Writing Sub Queries

Aim :- To perform advanced query processing and test its heuristics using correlated and nested subqueries, such as finding summary statistics.

4.1) : Retrieve all temple details, including the count of tickets booked for each temple.

Sql

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

Output:

TempleID	TempleName	Location	DeityName	TotalTicketBooked
T001	Golden Temple	Amritsar	Guru	5
T002	Tirupati	AP	Balaji	12
T003	Meenakshi	Madurai	Meenakshi	

4.2 :- Retrieve the total no. of cancelled bookings temple wise.

Sql

```
SELECT t.Name AS TempleName, COUNT(*) AS
TotalCancelled
FROM Temple t
JOIN Booking b ON t.TempleID = b.TempleID
WHERE b.status = 'Cancelled'
```

Output:

TempleName	TotalCancelled
Tirupati	2
Golden Temple	

4.3 :- Retrieve temple details where tickets have been booked.

```
Sql
SELECT *
FROM Temple
WHERE TempleID IN (
    SELECT DISTINCT TempleID
    FROM Booking
    WHERE Status = 'confirmed'
);
```

o/p:

TempleID	Name	Location	DeityName	Con_No
T001	GoldenTemple	Amritsar	Guru	9876543210
T002	Tirupati	AP	Balaji	

4.4 :- Retrieve visitor and booking details of visitors above 50 years old.

```
Sql
SELECT v.VisitorID, v.FullName, v.Age, b.BookingID,
b.BookingDate, b.Status,
FROM Visitor v.Booking b
WHERE v.VisitorID = b.VisitorID
AND v.VisitorID IN (
    SELECT VisitorID
    FROM Visitor
    WHERE Age > 50
);
```

o/p:

VisitorID	FullName	Age	BookingID	BookingDate	Status
V001	RameshKumar	55	B001	25-7-1	confirmed
V001	RameshKumar	55	B003	25-7-10	

4.5 :- Retrieve temples where no tickets have been booked.

Sql

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

O/P:-

TempleID	Name	Location	DeityName	Con-No
T003	Meenakshi	Madurai	Meenakshi	

4.6 :- Retrieve the TempleID, Name, Location, and visitor Name for a given VisitorID.

Sql

```
SELECT t.TempleID, t.Name AS TempleName, t.Location,
  v.FullName
FROM Temple t
JOIN Booking b ON t.TempleID = b.TempleID
JOIN Visitor v ON b.VisitorID = v.VisitorID
WHERE v.VisitorID = 'V002';
```

Output:-

TempleID	TempleName	Location	FullName
7002	Tirupati	AndhraPradesh	Sita Lakshmi

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

14/8

Result: Thus, the queries using joins and subqueries for the Online Temple Ticket Booking Management System have been executed successfully.

Task No:- 05 Date:- 21/8/25

Writing Join Queries, equivalent, and/or 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.TicketID, tk.Type,
       tk.Price
FROM Temple tm
JOIN Ticket tk ON tm.TempleID = tk.TempleID;
```

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

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

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

```
SELECT tm.Name AS TempleName,
COUNT(b.BookingID) AS TotalBookings
FROM Temple tm
LEFT JOIN Ticket tk ON tm.TempleID = tk.TempleID
LEFT JOIN Booking b ON tk.TicketID = b.TicketID
GROUP BY tm.Name;
```

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

```
SELECT d.DevoteeID, d.DevoteeName, d.ContactNo,
       tm.Name AS Temple
FROM Devotee d
JOIN Booking b ON d.DevoteeID = b.DevoteeID
JOIN Ticket tk ON b.TicketID = tk.TicketID
JOIN Temple tm ON tk.TempleID = tm.TempleID
WHERE tm.Name = 'Tirupati Balaji';
```

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

```
SELECT d.DevoteeID, d.DevoteeName, d.contactNo,
       COUNT(b.BookingID) AS TotalTickets
FROM Devotee d
LEFT JOIN Booking b ON d.DevoteeID = b.DevoteeID
GROUP BY d.DevoteeID, d.DevoteeName, d.contactNo
```

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

```
SELECT tm.Name AS Temple,
    COUNT(b.BookingID) AS SpecialDarshanCount
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
JOIN Ticket tk ON tm.TempleID = tk.TempleID
WHERE tk.AvailableSeats > 0;
```

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

```
SELECT d.DevoteeID, d.DevoteeName, d.Age, b.BookingID,
    b.BookingDate, tm.Name AS Temple
FROM Devotee d
JOIN Booking b ON d.DevoteeID = b.DevoteeID
JOIN Ticket tk ON b.TicketID = tk.TicketID
JOIN Temple tm ON tk.TempleID = tm.TempleID
WHERE d.Age > 40;
```

Outputs

5.1)

Temple	Ticket ID	Type	Price
Tirupati Balaji	T01	Special Darshan	300
Tirupati Balaji	T02	VIP Darshan	500
Madurai Meenakshi	T03	General	100
Rameshwaram Temple	T04	Special Entry	250

5.2)

Booking ID	Temple	Devotee	Visit Date
B001	Tirupathi	Ramesh	10-7-23
B002	Meenakshi	Priya	11-7-23

5.3)

Temple	Bookings
Tirupati	5
Meenakshi	3
Srirangam	2

5.4)

DevoteeID	Name	Contact No
D101	Ramesh	9876543210
D102	Kavitha	9543321654
D103		

5.5)

TempleID	TempleName	Status
T001	Tirupati	1
T002	Meenakshi	2

5.6)

TempleName	Cancelled
Tirupati	1
Srirangam	2

5.7)

TempleID	Name	Location	Deity Name
T001	Tirupati	Andhra	Balaji
T002	Meenakshi	Madurai	Meenakshi

5.8)

DID	Name	Age
D110	Suresh	55

5.9)

TID	Name	loc	Deity Name
T005	Palani	Dindigul	Murugan

5.10)

TID	DID	TName	DName
T002	D135	Meerakshi	Kavya

5.9) To retrieve temple details where no bookings 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 templeID, ticketID, temple name and devotee name for a particular bookingID 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
JOIN Devotee d ON b.DevoteeID = d.DevoteeID
WHERE b.BookingID = 'B002';
```

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	

21/8/25

Result: Thus, the queries using Join Queries, Nested Queries, and Equivalent Queries were executed successfully for the Online Temple Ticket BMS.

Task No:- 06 Date:- 26/8/25

Procedures, Functions and Loops

Aim:- To write PL/SQL Procedures, Functions and Loops on Ticket Booking and Managemt scenarios.

5.1) Write a PL/SQL block that calculates the average number of tickets booked per devotee and displays the result

```
DECLARE
    total_tickets NUMBER := 0;
    num_devotees NUMBER := 0;
    avg_tickets NUMBER := 0;
BEGIN
    -- loop through all devotees and sum their ticket counts
    FOR rec IN (SELECT COUNT(TicketID) AS ticket_count
                FROM TicketBooking
                GROUP BY DevoteeID) LOOP
        total_tickets := total_tickets + rec.ticket_count;
        num_devotees := num_devotees + 1;
    END LOOP;
    -- calculate average
    IF num_devotees > 0 THEN
        avg_tickets := total_tickets / num_devotees;
    END IF;

    DBMS_OUTPUT.PUT_LINE('Total Devotees: ' || num_devotees);
    DBMS_OUTPUT.PUT_LINE('Total Tickets: ' || total_tickets);
    DBMS_OUTPUT.PUT_LINE('Average Tickets per Devotee: ' ||
                         avg_tickets);
```

6.2) To write a PL/SQL block that inserts a new ticket booking record into TicketBooking table.

```
DECLARE
   v_TicketID VARCHAR2(6) := '&TicketID';
   v_DevoteeID VARCHAR2(6) := '&DevoteeID';
   v_TempleID VARCHAR2(6) := '&TempleID';
   v_BookingDate DATE :=
  TO_DATE('&BookingDate', 'YYYY-MM-DD');
   v_VisitDate DATE := TO_DATE('&VisitDate',
                        'YYYY-MM-DD');
   v_NoOfTickets NUMBER := &NoOfTickets;
   v_Amount NUMBER := &Amount;

BEGIN
   INSERT INTO TicketBooking (TicketID, DevoteeID,
   TempleID, BookingDate, VisitDate, NoOfTickets,
   Amount)
   VALUES
   COMMIT;
   DBMS_OUTPUT.PUT_LINE('Ticket booking
                record inserted successfully.');

EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error: '||
                        SQLERRM);

      ROLLBACK;

END;
```

6.3) Creak a function that returns the total number of bookings for a particular temple.

```
CREATE OR REPLACE FUNCTION
GetTotalBookingsForTemple (p_TempleID VARCHAR2)
RETURN NUMBER IS
   v_TotalBookings NUMBER:=0;
BEGIN
   SELECT COUNT (*) INTO v_TotalBookings
   FROM TicketBooking
   WHERE TempleID =p_TempleID;

   RETURN v_TotalBookings;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      RETURN 0;
   WHEN OTHERS THEN
      RETURN -1;
END GetTotalBookingsForTemple;
/
```

6.4) Write a non-recursive PL/SQL procedure to ~~retrieve~~ even-numbered TicketIDs.

```
CREATE OR REPLACE PROCEDURE
GetEvenNumberedTickets IS
BEGIN
   FOR rec IN (SELECT TicketID
               FROM TicketBooking
               WHERE
MOD(TO_NUMBER(SUBSTR(TicketID, 2))2) = 0)
LOOP
```

6.1) Output:-

Total Devotees: 10

Total Tickets: 45

Average Tickets per Devotee: 4.5

6.2) I/P ex :-

TicketID : 7102
DevoteeID : D501
TempleID : TMP01
Booking Date : 2025-09-03
Visit Date : 2025-09-10
NoOfTickets : 3
Amound : 150

O/P :-

Ticket booking record inserted successfully

6.3) execution:

o/p:

Total Bookings for Temple TMP01: 25

64)

exe

```
BEGIN
   GetEvenNumberedTickets;
END;
/
```

o/p:

Even-Numbered Ticket ID : T102

Even-Numbered Ticket ID : T204

Even-Numbered Ticket ID : T306

```
    DBMS_OUTPUT.PUT_LINE('Even-Numbered
Ticket ID: ' || rec.Ticket_ID);
    END LOOP;
END GetEvenNumberedTickets;
/
```

PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	

Result:- Thus, the PL/SQL Procedures, functions, and loops were successfully implemented for the Online Temple Ticket Booking Management System, and results were verified.

Task No: 07

Date: 4/9/25

Triggers, Views and Exceptions

Aim: To conduct triggers, views and exceptions on CRUD operations for restricting phenomenon in the Online Temple Ticket Booking Management System.

a) Create a trigger that automatically inserts a record in the Payment Table when a new booking is inserted into the TicketBooking Table.

```
CREATE OR REPLACE TRIGGER
insert_payment_record
AFTER INSERT ON TicketBooking
FOR EACH ROW
BEGIN
  INSERT INTO Payment (PaymentID, TicketID,
  Amount, PaymentStatus, PaymentDate)
  VALUES (
    'P' ||
  TO_CHAR(SYSDATE, 'YYYYMMDDHH24MISS'),
      :NEW.TicketID,
      :NEW.Amount,
      'pending',
      SYSDATE
    );
END;
/
```

b) View

Create a view that displays Devotee details along with their Booking and Temple information.

```
CREATE OR REPLACE VIEW
DevoteeBookingDetails AS
SELECT
    d.DevoteeID,
    d.FName || ' ' || d.LName AS DevoteeName,
    d.Email,
    t.TempleName,
    tb.TicketID,
    tb.BookingDate,
    tb.VisitDate,
    tb.NoOfTickets,
    tb.Amount
FROM Devotee d
JOIN TicketBooking tb ON d.DevoteeID =
                tb.DevoteeID
JOIN Temple t ON tb.TempleID = t.TempleID;
```

c) Procedure

```
CREATE OR REPLACE PROCEDURE
GetEventTicketIDsForTemple(
    in_TempleID VARCHAR2) AS

BEGIN
    FOR rec IN (
        SELECT TicketID
        FROM TicketBooking
        WHERE TempleID = in_TempleID
          AND
```

Outputs

7.a)
execution eg:

```
INSERT INTO TicketBooking
VALUES ('T105', 'D502', 'TMP01', SYSDATE, DATE
        '2025-09-15', 2, 200);
```

o/p:

1 row inserted into TicketBooking.
Triggers fired → New payment record inserted into Payment table.

o/p

7.b)

execution ex:

SELECT * FROM DevoteeBookingDetails;

o/p:

DevoteeID DevoteeName Email
TempleName TicketID BookingDate VisitDate
NoOfTickets Amount

D501 Rahulsharma rahul@gmail.com
Tirupati Balaji T101 01-SEP-25 10-SEP-25
2 200

7.c)

Execution ex:

```
BEGIN
    GetEventTicketIDsForTemple('TMP01');
END;
/
```

o/p:-

Even TicketID : T102
Even TicketID : T104

```
MOD (TO_NUMBER (SUBSTR(TicketID,2)),2):0)
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Even TicketID:'||
rec.TicketID);
  END LOOP;
END;
```

PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	+3
TOTAL (20)	(5+3=)8

Result: Thus, the Triggers, Views, and Procedures with Exceptions were successfully created and executed for Online ticket Booking System, and the results were verified

Task No:- 08 Date:- 11/9/24

CRUD operations in Document databases

Aim:- To perform Mongoose using NPM design on MongoDB for the Online Temple Ticket BMS, designing a document DB and performing CRUD operations like creating, inserting, querying, finding, updating, removing operations.

STEPS :-

STEP 1:- Install MongoDB

2) Install Mongosh
Download MongoDB Shell

3) Add the MongoDB 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 MongoDB shell binary location.

4) Confirm PATH by typing in Command prompt:
mongosh --help

5) Open MongoDB server
c:\programfiles\mongoDB\server\bin\mongod.exe

6) Perform CRUD Operations.

CRUD Operations for Online Temple Ticket, MMS

1) Create Collection

db.createCollection("TempleTickets")

{"ok":1}

2) Insert a Single Document

db.TempleTickets.insertOne({

TicketID: "T1001",

TempleName: "Meenakshi Amman Temple",

Location: "Madurai",

DevoteeName: "Ramesh Kumar",

BookingDate: "2025-09-10",

TicketPrice: 200

})

3) Find one Document

db.TempleTickets.find({TicketID: "T1002"})

4) Insert Multiple Documents

db.TempleTickets.insertMany([

{ TicketID: "T1002",

TempleName: "Brihadeeshwara Temple",

Location: "Thanjavur",

TicketPrice: 150

},

{ TicketID: "T1003",

TempleName: "Kapaleeshwarar Temple",

Location: "Chennai",

TicketPrice: 100

Output:

2) {
"acknowledged": true,
"insertedId":
ObjectId("651cf1726bbfe7993ddf999")
}

3) {
"_id": ObjectId("651cf1726ebbfe999"),
"TicketID": "TID01",
"TempleName": "Meenakshi Amman Temple",
"Location": "Madurai",
"BookingDate": "2025-09-10",
"TicketPrice": 200

4) {
"acknowledged": true,
"insertedIds": [
ObjectId("651cf1726ebbfe7993901"),
ObjectId("651cf1726ebbfe7993902"),
ObjectId("651cf1726ebbfe7993903")
]
}

5)

```
{
  "TicketID": "T1001",
  "TempleName": "Meenakshi Amman Temple",
  "Location": "Madurai",
  "TicketPrice": 200
}
```

6) {"acknowledged": true, "matchedCount": 1, "modifiedCount": 1}

7) {"acknowledged": true, "deletedCount": 1}

7)

5) Find All Documents

```
db.TempleTickets.find().pretty()
```

6) Update a Document.

```
db.TempleTickets.updateOne(
  {TicketID: "T1003"},
  {$set: {TicketPrice: 120, DevoteeName: "Vignesh kumar"}}
)
```

7) Delete a Document

```
db.TempleTickets.deleteOne({TicketID: "T1002"})
```

VEL TECH - CSE	
EX NO	8
PERFORMANCE (5)	4
RESULT AND ANALYSIS (5)	4
VIVA VOCE (5)	4
RECORD (5)	4
TOTAL (20)	4+4=18
SIGN WITH DATE	16.09.24

Result:- Thus CRUD operations (Create, Read, Update, Delete) were successfully performed using MongoDB with NPM/Mongoose design.

Task No: 09 Date: 18/9/25

CRUD operations in Graph database

Aim: To perform CRUD operations like creating inserting Querying, updating and deleting operations on graph spaces using neo4j Aura database for an online TTBMS.

Steps to get started with Neo4j Aura

1. Open Neo4j Aura
2. Click start free → continue with Google
3. Click open and download the credentials.txt file.
4. copy the password from the downloaded file when prompted.
5. Close beginner guides and start practicing Queries

CRUD operations

1. Create Temple Node

CREATE (t:Temple {temple ID: 'T001', Name: 'Sri Venkateswara Temple', location: 'Tirupathi', Phone: 9876543210})

Return t;

2. Create Ticket counter Nodes

CREATE (tc1:Ticket Counter {Counter ID: 'C001', Temple ID: 'T002', Type: 'Darshan', Price: 500})

Return tc1;

CREATE (tc2:Ticket counter {Counter ID: 'C002', Temple ID: 'T001', Type: 'special seva', Price: 1000})

Return tc2;

3) Create Devotee (USER) Nodes

CREATE (d1 : Devotee { DevoteeID: 'D01', Name: 'Arjun', Age: '30', Email: 'aryan@gmail.com'}) RETURN d1;

CREATE (d2: Devotee {DevoteeID: 'D02', Name: 'Meera', Age: 28, Email: 'meera@gmail.com'}) RETURN d1;

4) Create relationships

Temple → counters

Match (t: Temple {TempleID: 'T001'}),

(tc1: Ticket Counter {counterID: 'C001'})

CREATE (t) - [: Has-COUNTER] → (tc1) RETURN t, tc1;

Match (t: Temple {TempleID: 'T001'})

(tc2: Ticket Counter {counterID: 'C002'})

CREATE (t) - [: Has-COUNTER] → (tc2) RETURN t, tc2

Devotees → Booking

MATCH (d1: Devotee {DevoteeID: 'D01'}),

(b1: Booking {BookingID: 'B1001'})

CREATE (d1) - [: Booked] → (b1) RETURN d1, b1;

MATCH (d2: Devotee {DevoteeID: 'D02'}),

(b2: Booking {BookingID: 'B1002'})

CREATE (d2) - [Booked] → b2) RETURN d2, b2;

Bookings → Counters

MATCH (b1: Booking {BookingID: 'B1001'}),

(tc2: Ticket Counter {counterID: 'C002'})

CREATE (b1) - [: for-COUNTER] → (tc2) RETURN b1, tc2;

MATCH (b2: Booking {BookingID: 'B1002'}),

(tc1: Ticket counter {CounterID: 'C001'})

CREATE (b2) - [: for-COUNTER] → (tc1) RETURN b2, tc1;

Outputs :-

1) C:Temple {TempleID : "T001", Name: Sri Venkateswara Temple", Location : "Tirupathi", Phone: 9876543210}

2) C:Ticket Counter {CounterID : "C001", TempleID: "T001", Type: "Darshan", Price: 500})
C:Ticket Counter {CounterID : "C002", TempleID: "T001", Type: "Special Seva", Price: 1000})

3) C: Devotee {DevoteeID: "D01", Name: "Arjun", Age: 30, Email: "arjun@gmail.com"})
C: Devotee {DevoteeID: "D02", Name: "Meera", Age: 28, Email: "meera@gmail.com"})

4) C: Booking {BookingID: "B1001", Date: "25-09-20", Seats: 2, Amount: 1000})

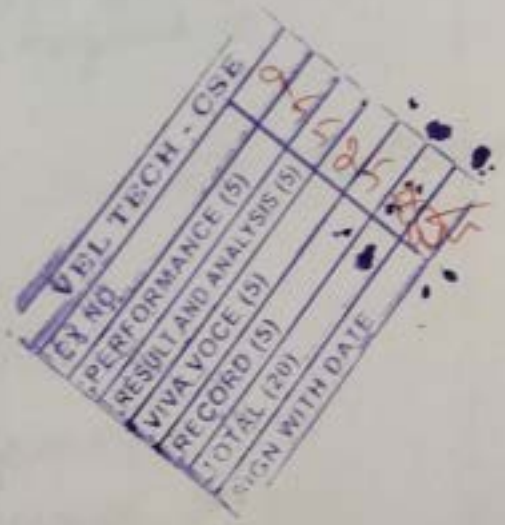
5) (Temple) - [:Has-COUNTER] → (Ticket Counters)
. (Devotee {D01}) - [:Booked] → (Booking {B1001})

6) Display All Nodes.
MATCH (n) RETURN n;

7) Retrieve Particular Devotee's Booking.
MATCH (d:Devotee {DevoteeID: 'D01'}) -[:BookedID]→
(b:Booking) RETURN d, b;

8) Update Booking Details.
MATCH (b: Booking {Booking ID: 'B1001'})
SET b.seats = 3, b.Amount = 1500 RETURN b;

9) Delete a Booking.
MATCH (b: Booking {Booking ID: 'B1002'})
DELETE b;



18/9

Result :- Thus the CRUD operations were successfully executed in Neo4j Aura Graph DB for an Online TTBMS.

Task No:- 10

Date: 25/9/25

Normalization of Online Temple Ticket Booking Management System

Aim:- To normalize the given relation of Online Temple Ticket Booking Management System and create simplified tables with suitable constraints up to Third Normal Form (3NF).

1) Identify Functional Dependencies (FDs)

i) User information
UserID → UName, UEmail, UPhone

ii) Temple information
TempleID → TName, Location

iii) Booking information
BookingID → UserID, TempleID, Date, TimeSlot, TicketCount, Amount, PaymentID.

iv) Payment information
PaymentID → PayDate, PayMode, PayStatus.

v) Priest information
PriestID → PName, PContact

vi) Special Pooja information.
SpecialPoojaID → SPName, SPFee

2) First Normal Form (1NF)

- Ensure atomic values, no repeating groups
- Given attributes are already atomic.

✓ Relation is in 1NF.

3) Second Normal form (2NF):-

. Must be in 1NF
. All non-prime attributes must depend fully on the candidate key.

Candidate Key : Booking ID

. Booking ID determines all details of a booking
. But some attributes depend only on their own entity's key, not on Booking ID.

We decompose into seperate relations:

1. User (UserID, UName, UEmail, UPhone).
2. Temple (TempleID, TName, Location).
3. Priest (PriestID, PName, PContact).
4. SpecialPooja (SpecialPoojaID, SPName, SPFee)
5. Payment (PaymentID, PayDate, PayMode)
6. Booking.

✓Now each table has attributes fully dependant on its primary key.
✓Relation is in 2NF.

4) Third Normal form (3NF):-

. Must be in 2NF
. No transitive dependencies.

Check FDs:

. UserID → UName, Utmail, UPhone (All)
. templeID → tName, Location (All)
. PaymentID → PayDate, PayMode, PayStatus (All)
. PriestID → pName, pContact (All)
. SpecialPoojaID → SPName, SPfee (All)
. BookingID → UserID, templeID ... (All)

No transitive dependencies.

√ Relation is in 3NF.

5) Minimal / Canonical Cover of FDs

Minimal cover:

. Remove redundant attributes from left side

. Canonical cover is same set as given

UserID → UName, Utmail, UPhone

TempleID → TName, Location

BookingID → UserID, TempleID, Date, Timeslot, TicketCount, Amount, PaymentID, PriestID, SpecialPoojaID

PaymentID → PayDate, PayMode, PayStatus.

PriestID → PName, PContact.

SpecialPoojaID → SPName, SPfee

Final Simplified Tables (3NF)

1. User

```
CREATE TABLE User (
    UserID INT PRIMARY KEY,
    UName VARCHAR(100),
    UEmail VARCHAR(100) UNIQUE,
    UPhone VARCHAR(15) UNIQUE
);
```

2. Temple

```
CREATE TABLE Temple (
    TempleID INT PRIMARY KEY,
    TName VARCHAR(100),
    Location VARCHAR(100)
);
```

3. Priest

```
CREATE TABLE Priest (
    PriestID INT PRIMARY KEY,
    PName VARCHAR(100),
    Pcontact VARCHAR(15)
);
```

4. CREATE TABLE SpecialPooja (

```
    SpecialPoojaID INT PRIMARY KEY,
    SPName VARCHAR(100),
    SPfee DECIMAL(10,2)
);
```

Outputs

1)

UserID	UName	UEmail	UPhone
1	Ravikumar	ravi@gmail.com	[illegible]
2	Anjali Sharma	anjali@gmail.com	[illegible]

2)

TempleID	TName	Location
101	Tirupati Balaji	Tirupati
102	Meenakshi Amman	Madurai

3)

PriestID	PName	PContact
201	Pandit Sharma	9123456780
202	Swami Iyer	9988776655

4)

SpecialPoojaID	SPName	SPFee
301	Abhishekam	500.00
302	Archana	300.00

5)

PaymentID	PayDate	PayMode	PayStatus
401	25/9/25	UPI	Success
402	25/9/25	Card	Pending

5. Payment

```
CREATE TABLE Payment (
    PaymentID INT PRIMARY KEY,
    PayDate DATE,
    PayMode VARCHAR (50),
    PayStatus VARCHAR (50)
);
```

VEL TECH - CSE	
EX NO	10
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	19
SIGN WITH DATE	

25/9

Result: Thus, the given Online TTBMS relation is successfully normalized up to Third Normal Form (3NF) and decomposed into simplified relations with suitable primary key, foreign key and uniquener

Task No: 11

Date :- 41

Menus, Forms and Reports 9/10/25

Aim: To design an Online Temple Ticket Booking MS application using Oracle forms, Menus, and Report Builder, as

Install Oracle Forms and Report Builder, Ensure that Oracle Forms Builder and Oracle Report Builder are properly installed and configured on your development system.

Design the Data Model

In Oracle Forms, define the data model that connects to the DB schema for the temple booking system.

Use the Data Block Wizard to create blocks

- Devotee (ID, Name, Address, Ph, Email)
- Temple (ID, Name, Location, Timing).
- Booking (ID, Date, Type, Amnt, Payment Status)
- Admin (ID, Username, Password)

Create Menus

Menus provide easy navigation for diff operations like booking management, viewing reports, and user settings.

Steps to create menus in Oracle forms:

1) Open Oracle Forms Builder.
2) Create a new form module.
3) Add menu items such as:
   - Create Management → Add Flights / Schedule
   - Manage Details → View Flights / Update Info
   - Ticket Booking → New Booking / Cancel
   - Reports → Booking Reports / Payment
   - Logout / Exit
4) Define hierarchy and assign triggers or PL/SQL procedures for each menu action.
5) Compile and run to test.

Design forms

1) Create new forms
2) Add form elements such as text fields, combo boxes, buttons and labels
3) Use the property Palette to configure form element properties.
4) Write PL/SQL code for business logic such as:

. Validating devotee info.
. Checking ticket availability
. Processing booking and Payment.

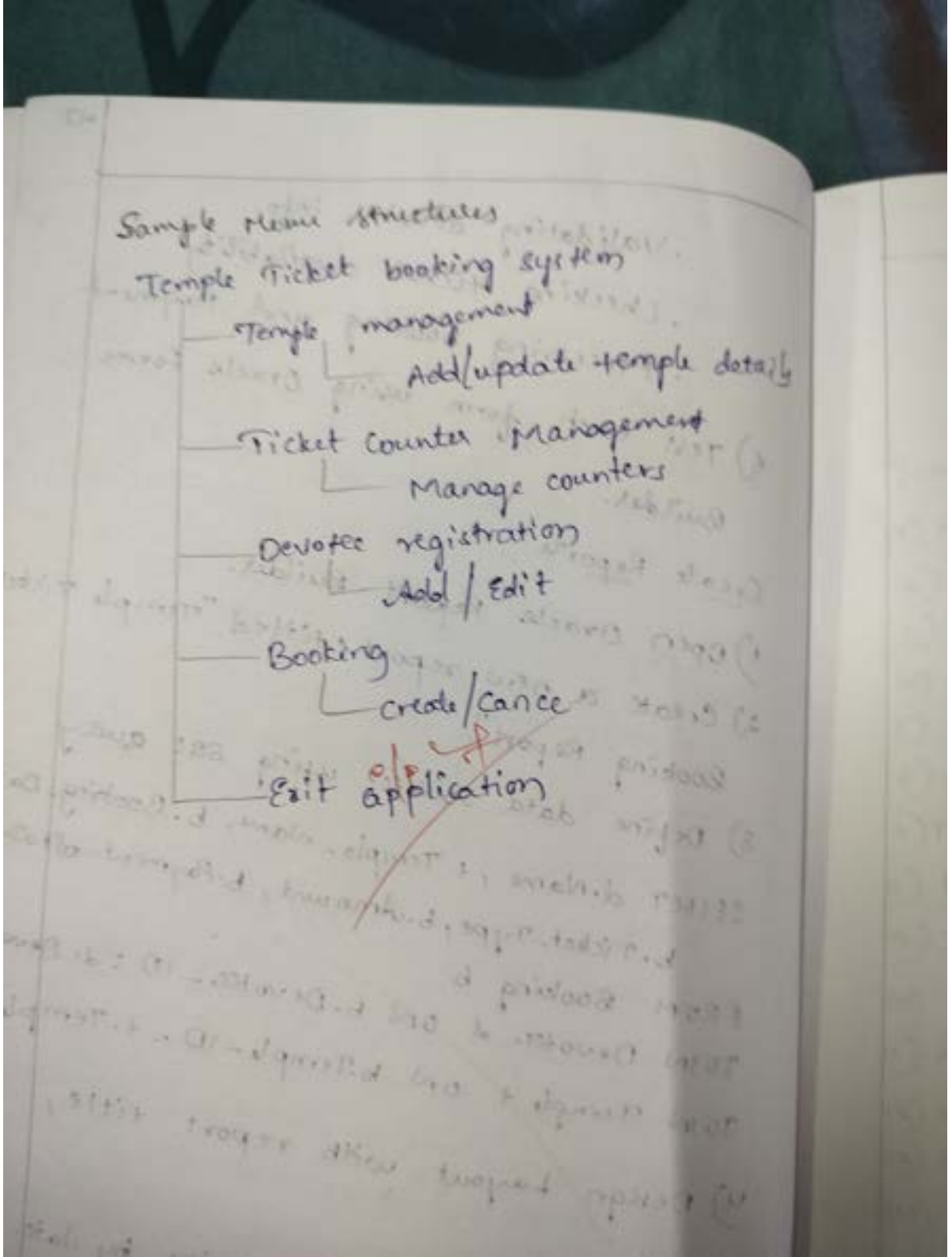
5) Test each form using Oracle Forms Builder.

Create Reports

1) Open Oracle Report Builder.
2) Create a new report titled "Temple Ticket Booking Report".
3) Define data source using SQL query

```
SELECT d.Name, t.Temple_Name, b.Booking_Dat,
    b.Ticket_Type, b.Amount, b.Payment_Status
FROM Booking b
JOIN Devotee d ON b.Devotee_ID = d.Devotee_ID
JOIN Temple t ON b.Temple_ID = t.Temple_ID;
```

4) Design Layout with report title, headers.
5) Add parameters for filtering by date.
6) Preview and format the report to ensure all fields display correctly.


Sample Menu structures

Temple Ticket booking system

- Temple management
  - Add/update temple details
- Ticket Counter Management
  - Manage counters
- Devotee registration
  - Add / Edit
- Booking
  - create/cance
- Exit application

VEL TECH - CSE	
EX NO	
PERFORMANCE (5)	
RESULT AND [illegible] (5)	
VIVA VOC	
RECORD	
TOTAL (20)	
SIGN WITH DATE	

09.10.24

Result: Thus, the Online Temple Ticket BMS was successfully designed using Oracle Forms, Menus, and Report Builder.

Use Case :- 04

Date:-16/10/25

Army Supply Chain, Bill of Matairals and Maintenance Cost Management.

Team Members :-

1) M. RamaTulasi
2) T. Bindu
3) G. Yasaswini
4) D. Pranathi
5) P. Hasika
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 analyzes 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 200,00 civilian staff. Each personnel 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 for various 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 Headquarters
- 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.

Required Outputs:

- Graphical dashboards showing real time data and historical values.
- Customized reports and maintenance schedules.
- Comparative reports on deployment and maintenance costs under different scenarios.

Postconditions:

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

Use Case Flow:-

step	Description
1	The Defense 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 part 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 operations.

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	

16/10/25

Result:- Thus, the Army Supply Chain and Maintenance Cost MS provides a robust analytical solution.