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: V. Vipin Raj

VTU.NO: 29362

REG.NO: 24UECL0063

BRANCH: CSE (AI & ML)

YEAR/SEM: 2nd year / III

SLOT: S10 L42

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

ACADEMIC YEAR 2025-26 (SUMMER SEMESTER)

BONAFIDE CERTIFICATE

NAME : Vavipin Raj
VTU NO. : 29362
YEAR/SEM : 2nd / III

BRANCH : CSE(AI & ML)
REG.NO. : 24UEC10063
SLOT NO. : S10+12

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	12	18	7/8
4.	14/08/25	Developing queries with DML Multi-row functions and operations	16	18	14/8
5.	21/08/25	Writing Join Queries, equivalent, and/or recursive queries	20	17	21/8
6.	28/08/25	Writing PL/SQL using Procedures, Function	24	19	28/8
7.	04/09/25	Writing PL/SQL using Loops	27	19	4/9
8.	11/09/25	Normalizing databases using functional dependencies up to BCNF	31	20	11/9
9.	18/09/25	Backing up and recovery in databases	34	20	18/9
10.	25/09/25	CRUD operations in Document databases	37	17	25/9
11.	09/10/25	CRUD operations in Graph databases	41	20	9/10
12.	16/10/25	Micro Project: A gift coupon application that handles offers and payment related information.	48	18	16/10

Total Marks: 222/240

Completed

R.T.T Yadav

Signature of Faculty

Record Mark

Temple ticket online booking Management system

S.NO	Date	Title	Marks	Sign
1.	24/07/25	concept Design after full text review.	13	24/7
2.	31/07/25	Generating design of a other traditional data base model.	13	31/07
3.	07/08/25	using clauses, operators & functions in queries.	13	7/8
4.	14/08/25	using functions in Queries & writing Sub Queries.	13	14/8
5.	21-08-25	writting Join queries equivalent & recursive queries.	12	21/8
6.	28/08/25	procedures Functions and loops	14	28/8
7.	4/09/25	Triggers, views and exceptions	14	4/9
8.	11/09/25	Crud operations in document data bases.	15	T, M, Th 16.09.25
9.	18/09/25	Crud operations in graph databases		
10.	25/09/25	Normalizing databases using functional dependencies upto third Normal form.		

Date. 24-07-2025

Conceptual Design after ITR

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:

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

Attributes:

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

Relationship:

An association or interaction between two or more entities, for example, a student might enroll

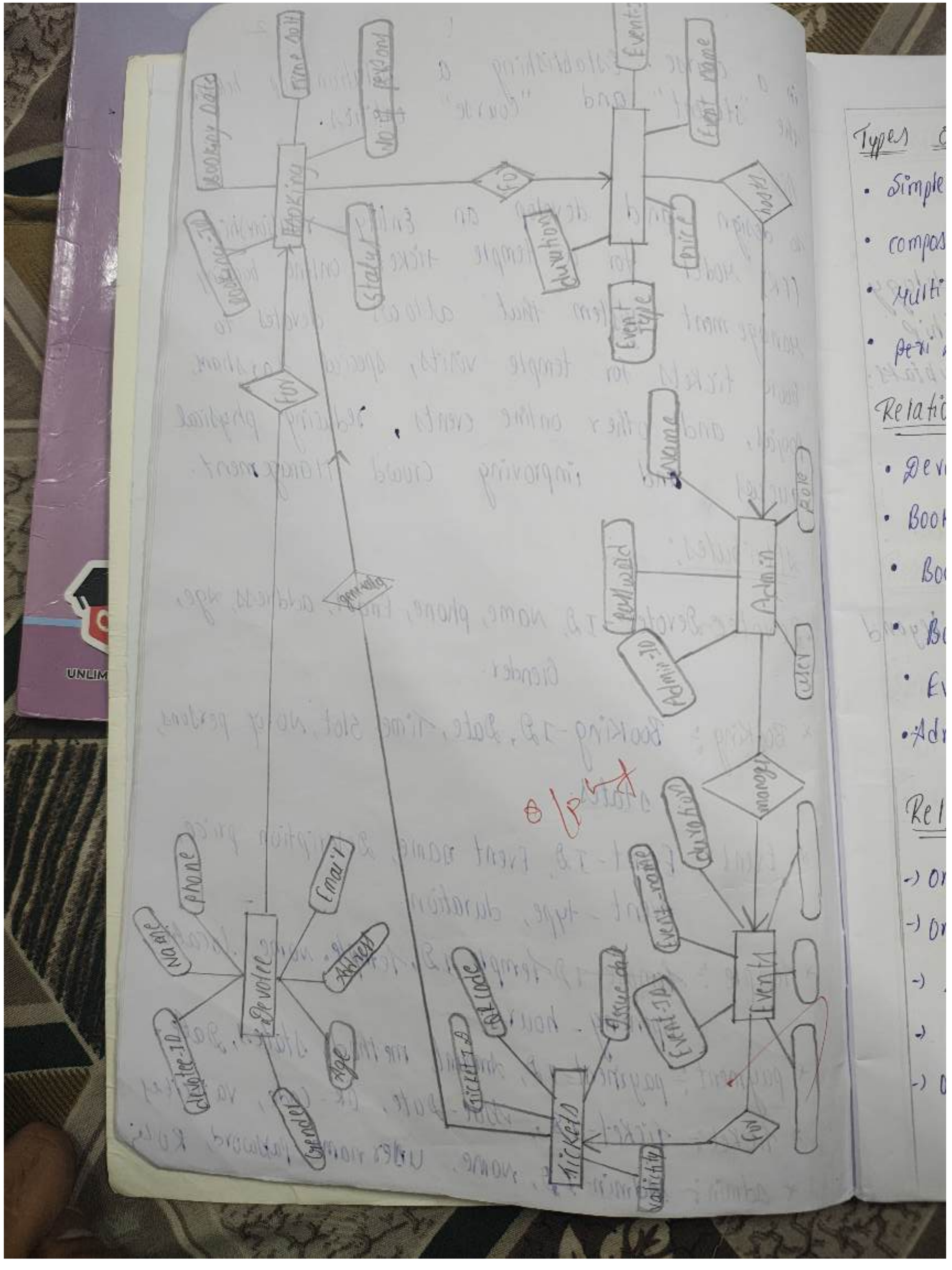
in a course. Establishing a relationship between the "student" and "course" entities.

Aim:

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

Attributes:

* Devotee: Devotee - ID, Name, phone, Email, address, age, Grender.
* Booking: Booking - ID, Date, Time - Slot, No. of persons, states.
* Event: Event - ID, Event name, Description price, event - type, duration.
* Temple: ~~Temple - ID~~ Temple - ID, Temple. name, location, opening - hours.
* payment: payment - ID, Amount, method, states, Date.
* Ticket: Ticket - ID, Issue - Date, QR - Code, validites
* Admin: Admin - ID, Name, Username, password, Role.


Booking Date
Booking
Status
Event
Event Name
Event Type
Duration
Prize
Name
Role
Admin
Admin-ID
Manage
Provider
Provider-ID
Phone
Email
Address
Age
Gender
Ticketed
Ticket-ID
Issue-date
Event-name
Event-ID
Duration
Event
Fee
Validity
1:N
1:1
M:N

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 → Ticket
-> one-to many : Devotee → Booking
-> Many-to-one : Booking → Event
-> one-to-Money : Temple → Event
-> one-to-Many : Admin → Event

Task-1: Conceptual Design after FTR.

FTR: Full text review.

CO1, K3

Using basic database design methodology and ER model, design entity relationship diagram by satisfying the following subtasks.

1. (a) identifying the entities
2. (b) identifying the attributes
3. (c) identifying of relationship. cardinality, type of relationship.
4. (d) Refraining the relations with beyond constraints.

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)	2
RECORD (5)	-
TOTAL (20)	13
SIGN WITH DATE	

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: 31/07/2025

5

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 hierarchy! has-a hierarchy & performs generalization and/or specialization relationship.

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

2d. Rename the relations

2e. Perform x 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)

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

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

Relation ship 3:- Customer books ticket (many-to-many).

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

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

Relationship 4:- Tickets corresponds to a specific seat, and each seat can be assigned to only one ticket.

Specificity:- Tickets allocated to seat (one-to-one)

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

26. 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 super class: user with above attributes

-> subclasses:

customer: Additional attributes like customer-ID, address, payment_INfo

Admin: Additional attributes like Admin_ID, Role

Specialization Example:-

-> Entity: Ticket

-> specialized into:

General Ticket (attributes: price, general_access_area)

vip Ticket (attributes: price, access_to_special area, complimentary_services)

27. C find the domain of attributes and perform check constraints.

Attribute	Domain	check constraint example
age (customer)	positive integer (min 18)	CHECK (age >= 18)
ticket-price	positive decimal number	CHECK (ticket_price > 0)
Booking-date	Date (not in Rust)	CHECK (booking_date >= current_date)
Seat-number	string or number within valid range	CHECK (seat_number btw 1 and 500);

2nd Rename the relations

Example renaming columns for clarity:

sql

ALTER TABLE customer Rename column Contact_no to phone no;

ALTER TABLE Ticket Rename column price to ticket_prince;

ALTER TABLE Booking Rename column book_date to Booking_date;

DCL (Data control language);

sql

-- Creating a new user and granting privileges

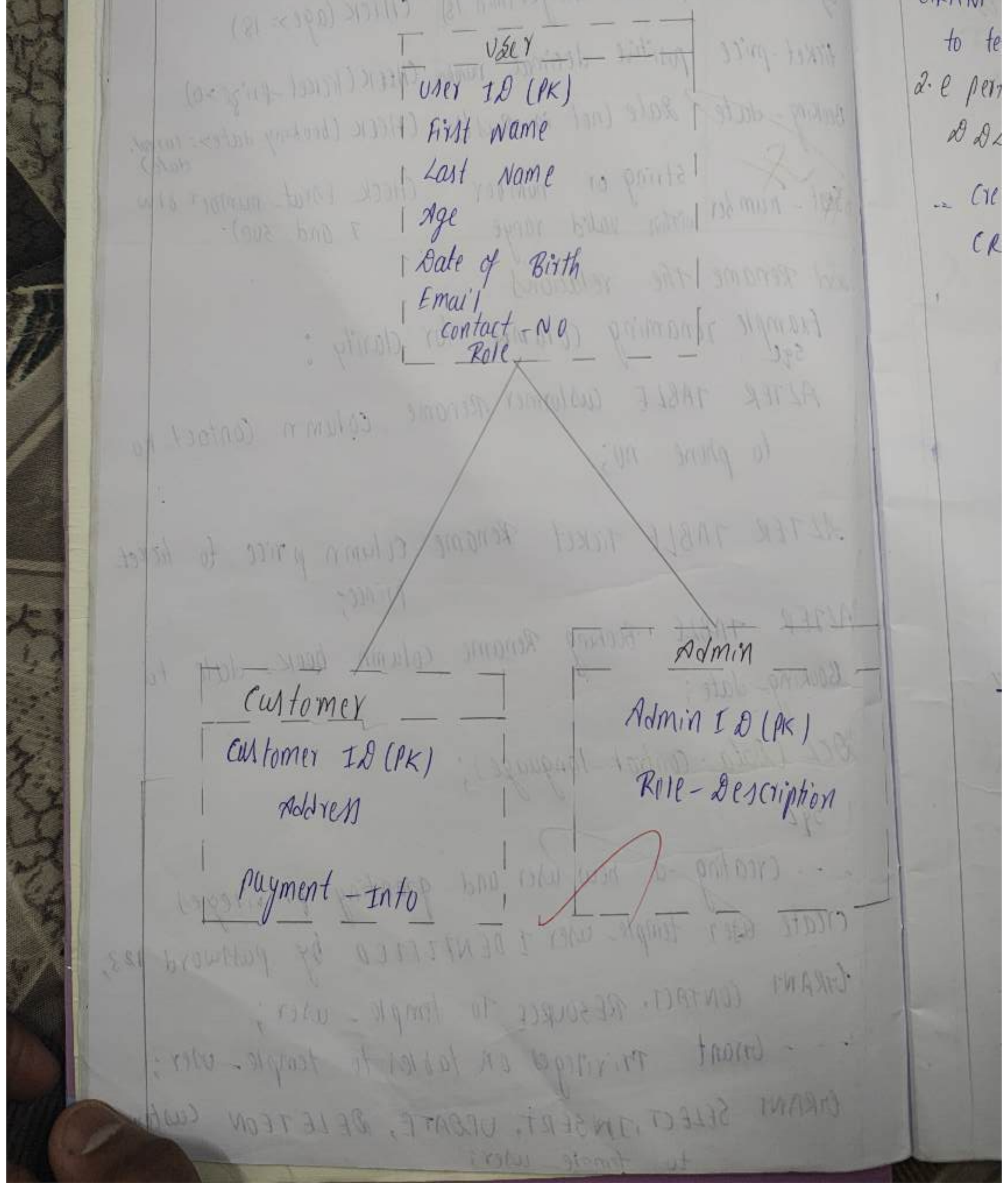
create user temple_user IDENTIFIED by password 123;

GRANT CONTACT, RESOURCE To temple_user;

--- Grant Privileges on tables to temple_user;

GRANT SELECT, INSERT, UPDATE, DELETEON customer to temple_user;

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



Event to temple_user;
GRANT SELECT, INSERT, UPDATE, DELETE ON Temple
to temple_user;

2. e perform SQL Relations using DDL & DCL commands
DDL (Data definition Language):

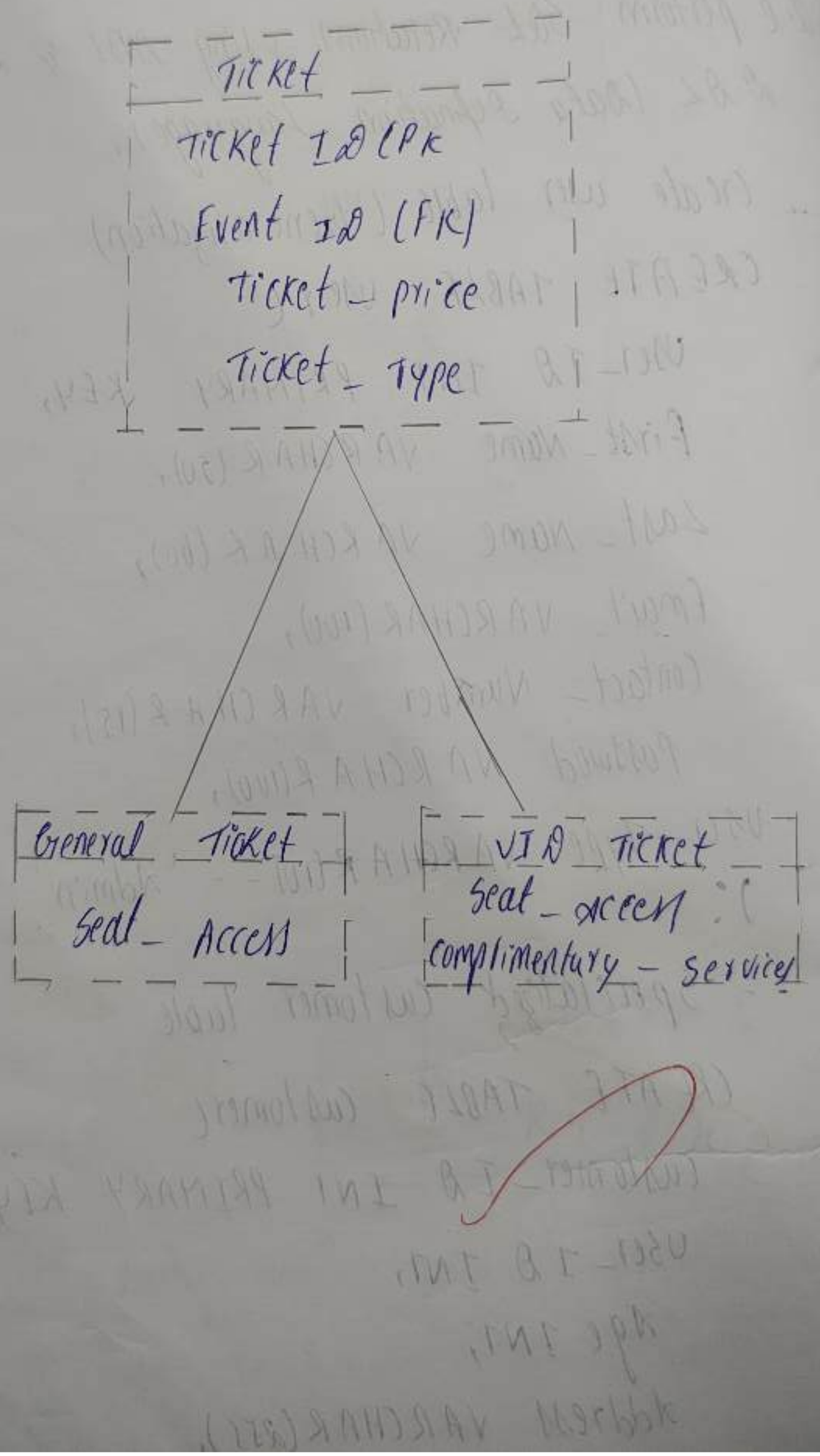
-- Create user table (Generalization)

```
CREATE TABLE User (
    User_ID INT PRIMARY KEY,
    First_Name VARCHAR(50),
    Last_Name VARCHAR(60),
    Email VARCHAR(100),
    Contact_Number VARCHAR(15),
    Passwod VARCHAR(1000),
User_Type VARCHAR(10) -- 'Admin' or 'Customer'
);
```

-- Specialized Customer table

```
CREATE TABLE Customer(
    Customer_ID INT PRIMARY KEY,
    User_ID INT,
    Age INT,
    Address VARCHAR(255),
    Payment_Info VARCHAR(255),
    FOREIGN KEY (User_ID)
    REFERENCES User (User_ID)
```

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



```
);

-- 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),
    Start_Date DATE,
    End_Date DATE,
    FOREIGN KEY (Temple_ID)
    REFERENCES Temple (Temple_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	4
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	

3/10/24

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 & DCL commands.

Date: 7-08-25

TASK-3: Using clauses, operators & functions in queries.

Aim:- To perform query processing on a temple ticket online Booking management system for different retrival 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.
T1001	Tirumatee venkatesh wara	Tirupati	9024276954
T1002	meenakshi Amman	Madurai	7989222725
T1003	Kashi vishwanth	varanari	7013982716
T1004	Jagannath Temple	Puri	9701224404
T1005	Golden Temple	Amritsar	9908616499

Visitor:

Visitor ID	FName	L Name	age	Email	Contact-NO
V001	Anil	Kumar	30	Anik@gmail.com	9014276954
V002	Akash	Reddy	25	Akashr@gmail.com	9701224404
V003	Priya	Sharma	28	Priyas@gmail.com	9908290010
V004	Aarav	Patel	35	Aaravp@gmail.com	7013982716
			22	Sneharp@gmail.com	7989222725

Booking:

Booking ID	Temple ID	visitor ID	Booking_date	Ticket Type	amount
B001	T1D01	V001	2024-06-15	VIP	500
B002	T1D02	V002	2024-06-16	General	100
B003	T1D01	V003	2024-06-11	General	100
B004	T1D03	V004	2024-06-18	VIP	100
B005	T1D01	V005	2024-06-19	General	200

Priest:

Priest ID	F Name	L Name	Age	Email	ContactNo
P001	Ramesh	Iyer	50	ramesh@gmail.com	9044276954
P002	Suresh	Sharma	45	Suresh@gmail.com	9501229404
P003	Manish	Pal	40	Manish@gmail.com	7989922278

2. Retrieve details of visitors whole first_name starts with 'n'.

```
SELECT*
FROM visitor
WHERE FNAME CRKE 'A%';
```

Results:

Visitor ID	FName	LName	Age	contact. NO
V002	Akash	Reddy	25	9701224401
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 TEMPLE ID IN('TID01', TID03, TID04');
```

Result:-

Temple ID	Name	location	contact No
TID01	Trimula	~~Tirj~~ Tirupathi	9701224404
TID02	Kashi	varanasi	7013952716
TID03	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
WHERE Temple ID is NULL;
```

Result:

(No result if all priests are assigned)

VEL TECH - CSE	
EX NO	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	13
TOTAL (20)	
SIGN WITH DATE	

#18

Result:

Thus, query processing for temple ticket online booking management system using clauses, operation, and functions was successfully performed.

Date: 14-08-2025 16

Task-4: Using functions in queries and writing Subqueries:

Aim: To perform advanced query processing and test its heuristics by designing optimal correlated and nested (sub quires such as finding summary statics in the temple tickets online 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 Temple Name
       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;
```

out put:

Temple ID	Temple Name	Location	Total Booking
T1001	Tirumala	Tirupati	3
T1002	Meenakshi Annam	madurai	1
T1003	Kashi	varanasi	1
T1004	Jagannath temple	puri	0
T1005	Golden temple	Amritsar	0

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

```
SELECT t.name AS Temple Name,
COUNT(*) AS total Special Bookings
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 when Bookings include 'VIP' tickets.

```
SELECT *
FROM Temple
WHERE Temple ID IN
   SELECT Temple ID
FROM Booking
WHERE Ticket-type = 'VIP'.
);
```

Output:

Temple ID	Name	location	Contact-No
TI201	Tirumala	Tirupathi	9701224404
TI203	Kashi	Varanasi	7013982716

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

```
SELECT v.visitor ID.
       v.Name AS visitor Name,
       v.age
       b.Booking ID,
       b.Booking_Date,
       b.ticket_Type,
       b.amount
FROM visitor v
JOIN Booking b
ON v.visitor ID = b.visitor ID
WHERE v.Age = 25;
```

Output:

visitor	visitorName	Age	BookingID	Amount
V001	Anil	30	B001	500
V003	Priya	28	B003	700
V004	Aarav	25	B004	700

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:

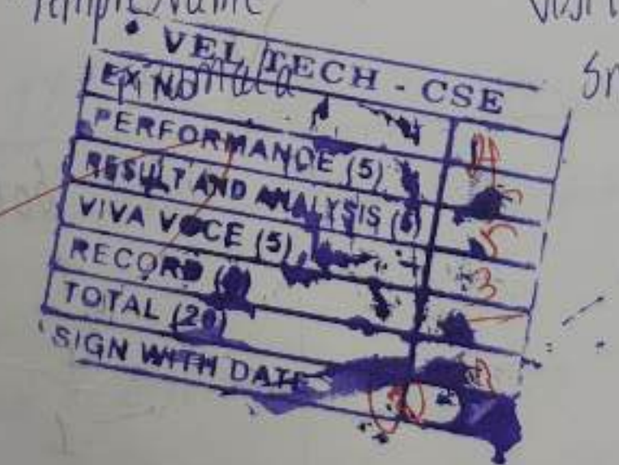
Temple ID	Name	location	Contact-NO
T1004	Jagannath temple	Puri	9701224404
T1005	Golden temple	Amristar	9014216954.

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

```
SELECT t.templeID,
       t.Name AS TempleName,
       v.FName AS visitorName
FROM temple t.
JOIN Booking b
ON t.templeID = b.templeID
JOIN visitor v
ON b.visitorID = v.visitorID
WHERE v.visitorID = 'V005';
```

Output:

TempleID	TempleName	visitorName
T1001		Sneha



Result: Thus, the queries using function, joins and nested sub queries were successfully executed for the temple ticket online Booking Management System.

Out Put :

Temple	pooja Name	price
Tirupati Temple	Suprabhata seva	300
Tirupati Temple	Archana	100
Kanchi Temple	Abhishekam	500
Madurai Temple	special Darshan	250

Out put :

Booking ID	Devotee	pooja name	Booking date	Slot time	Status
B101	Rajesh	Suprabhata seva	22-jun-23	6:00 AM	confirmed
B102	Meena	Archana	22-jun-23	9:00 AM	Confirmed
B103	Arjun	Abhishekham	23-jun-23	7:30 AM	confirmed
B104	Kavitha	special Darshan	23-jun-23	8:30 AM	Confirmed
B105	Suresh	Abhishekham	24-jun-23	6:30 AM	confirmed

Out put :

Devotee	Total bookings
Rajesh	1
meena	1
Arjun	1
kavitha	1
Suresh	1

Date: 21-08-2025.

Task-5: writing join Queries, equivalent, and for recursive Queries:

Aim: To perform advanced query processing and test its heuristics using join Queries, equivalent queries, and recursive queries from the temple ticket online booking management system, which manages devotees, priests, pooja, tickets and booking details.

Queries:

5.1 To retrive all temples and their poojas

Sql

SELECT t.Name as temple, p.Pooja Name, p.price
FROM Temple t
JOIN POOJa on t.temple ID = p.temple ID;

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

Sql

SELECT b.booking ID, d.t Name As Devotee, p.Pooja Name, b.booking Date, b.slot time, b.status
from booking b
JOIN devotee do N b.devotee = d.devotee ID
JOIN pooja on b.Pooja ID = p.Pooja ID.

5.3 count the number of bookings made by each devotee.

Devotee ID	F Name	Mobile No
D103	Arjun	9876543210
D105	Suresh	9876543210

output:

Pooja Name	Total Bookings
Suprabhota Seva	1
Archana	1
Abhishekam	1
special Darshan	1

Out put:

Temple	Cancelled Booking
Kanchi Temple	1

out put:

Temple ID	Name	location	Head priest
T001	Tirupat Temple	Tirupati	Ramanujam
T002	Kanchi Temple	Kanchipuram	Narayan Das
T003	Madurai Temple	Madurai	Sub ramainah

Sql

SELECT d.FName AS Devotee, Count (b.booking ID) AS total bookings

FROM Devotee d

LEFT JOIN Booking b on d. Devotee = b.Devotee ID

GROUP By d.FName

5.4 To find all devotes who booked 'Abhisheham'

Sql

SELECT d.devotee ID, d.FName, d. Mobile NO

FROM Devoteed.

JOIN Booking b on d. Devotee ID = b.devotee ID

JOIN pooja P ON b. pooja ID = P. pooja ID

WHERE P. pooja Name = 'Abhishekam'

5.5 To retrive all poojas and the number of times they were booked.

Sql

SELECT P.Pooja Name, Count (b. booking ID) AS Total Bookings

LEFT JOIN Booking b on P. Pooja ID = b. pooja ID

Group By P. pooja Name;

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

Sql

SELECT t. Name As Temple, COUNT (b. Booking ID) AS

JOIN Booking b. on. t. temple ID = b. temple ID

Output:

F Name	Age	Slot time	Booking ID	booking date	status
Suresh	55	6:30AM	B105	24-june-23	confirm

Output:

Temple ID	Name	location	Head Point
1004	Rameswaram temple	Rameshwaram	vishnu devon

Output:

Temple	pooja Name	Devotee	Booking date	Slot time
Tirupati temple	Archana	meena	22-jun-23	9:00 AM

WHERE b.Status = 'cancelled'
GROUP By E.Name,

5.7 To retrive temple t.temple ID, t.Name, t.location, were successfully.

Sql

SELECT DISTINCT t.temple ID, t.Name, t.location, t.Head priest
JOIN Booking b on t.temple ID = b.Temple ID
WHERE b.Status = 'confirmed'.

5.8 To retrive devotee and Booking details for devotees above 50 years old.

Sql

SELECT d.f Name, d.Age, b.Booking ID, b.booking Date, B.Slot time, b.Status

JOIN Booking b ON d.Devotee ID = b.devotee ID
WHERE d.Age > 50;

5.9 To retrive the details of temples where no pooja booking are dow.

SELECT *
FROM Temple
WHERE Temple ID NOT IN (SELECT Temple ID from booking);

5.10 To retrive temple, pooja and devotee detail for a give Booking ID.

Sql

SELECT t.Name AS temple, P.pooja Name, df.Name AS

```
FROM Booking b
JOIN Temple t on b.Temple ID = t.temple.ID
JOIN ~~Devotee~~ pooja p ON b.pooja = p.pooja.ID
JOIN Devotee d ON b.Devotee ID = d.Devotee ID
where b.booking ID = 'B02';
```

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

Result: Thus the temple ticket online booking management system using join queries, equivalent queries, and recursive, successfully to manage temple, devotees, priests, poojas and booking.

Date: 28/08/2025

TASK-6

Procedures, functions, and loop using PSL/SQL on number theory and bussiness scenarios.

Aim: To write and execute PSL/SQL procedure, function and loops on number theory (eg: Prime number, factorial) and bussiness scenarios (e.g., salary calculation, Discount calculation).

Software used: Oracle SQL / PL/SQL

Algorithm:

(A) Number theory Example - prime number check using procedure

1. Start
2. Read a number n.
3. Initilaze counter
4. use loop to check divisiblity from 2 to n/2.
5. If divisible → Not prime, else prime
6. Display result.

(B) Bussiness scenario Example - Salary Bonus Calculation using function.

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

Programs:

1) Number theory- prime number check (procedure with loop)

```
SET SERVER OUTPUT ON;
CREATE OR REPLACE PROCEDURE
check-prime (n IN number) is
   flag BOOLEAN := TRUE;
BEGIN
   IF n<=1 THEN
   DBMS-OUTPUT.PUT.line (n||'is NOT prime:);
ELSE
  FOR i in 2. n/2 loop
     if MOD (n,i)=0 then
        flag = FLASE;
        EXIT;
     END IF;
     END LOOP;
IF flag THEN
 DBMS_OUTPUT.PUT_LINE (n||" is prime");
ELSE
 DBMS_OUTPUT. PUT.LINE (n||' is NOT Prime');
 END IF;
 END IF;
 END;
/
 Execution
BEGIN
    check_prime (29);
    Check_prime (20);
END;
/
```

Out put:

For prime Number check:

29 is prime

20 is NOT prime

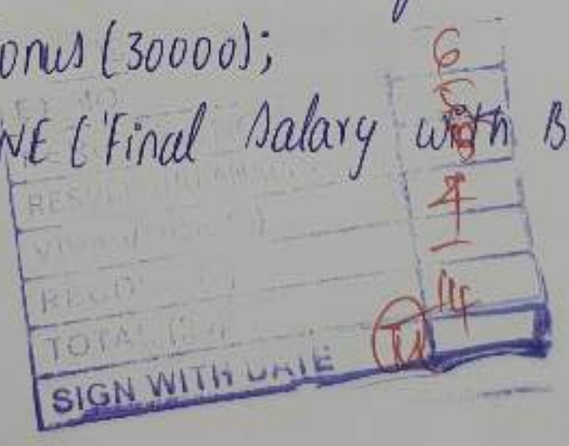
For Bonus Calculation:

Final salary with Bonus : 21600

Final salary with Bonus : 33000

(B) Bussiness scenario. Employee Bonus calculation (Function)

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



Result:

Thus, PL/SQL procedures, functions, and loops were successfully implemented for number theory and bussiness scenarios.

TASK-7

Triggers, Views and Exceptions

Aim: To conduct events, views, and exception handling on CRUD operations from the temple ticket online booking management system.

a) Trigger:

Requirment: when a new booking is inserted into the booking table. Automatically insert a corresponding record into the ticket table with default status as 'ACTIVE'.

sql

```
CREATE OR REPLACE TRIGGER
  trg_insert_ticket
  AFTER INSERT ON Booking
  FOR EACH ROW
BEGIN
  INSERT INTO Ticket (Ticket ID, Booking ID, Devotee Name, Age,
        Gender, QRCode, Status)
    VALUES (
    seq_ticket.NEXTVAL, --assuming a
  Sequence for ticket IDs
    :NEW.Booking ID
    'Unknown'. --Default place holder.
    'NULL' --Age not given at booking time
    Sys.GUID(). --auto-generate QR Code.
    'ACTIVE' --default ticket status
    );
  END;
  /
```

Sample Output:

BOOKING_ID	BOOKING_REF	DEVOTEE	TEMPLE	DARSHAN_TYPE	SLOT_START	SLOT_END
101	BKC2023001	Ramesh Kumar	Tirumala temple	General	2023-11-15 06:00:00	2023-11-15 07:00:00
102	BKC2023002	Priya sharma	Kathi Vishwanth	VIP	2023-11-16 08:00:00	2023-11-16 09:00:00
103	BKC2023003	Amit patel	Siddhivinyak	Special	2023-11-17 10:00:00	2023-11-17 11:00:00

B) View

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

Sql

```
CREATE OR REPLACE VIEW Booking Details view
AS
SELECT
    b. Booking ID,
    b. Booking Ref,
    u. Name AS Devotee Name,
    t. Name AS Temple Name,
    dt. Name AS DarshanType,
    s. Start Ts AS Slot start,
    S. End Ts AS Slot END,
    b. QTY,
    b. Amount,
    b. Status
FROM Booking b
JOIN Users U ON b. user ID = u. user ID
JOIN Temple t ON b. Temple ID = t. temple ID
JOIN Slot S ON b. slot ID = S. Slot ID
JOIN Darshan TYPe dt ON S. Darshan Type ID = dt. Darshan
                                              Type ID;
```

Sql

```
-- To query
    SELECT * FROM Booking Details view:
```

(C) Non-Recursive PL/SQL Procedure:

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

SQL

```
CREATE OR REPLACE PROCEDURE
Get Even Booking IDs For example(
        in_temple_id Number
        Out_even_booking-ids Out
  SYS.ODCI NUMBER LIST
  )AS
BEGIN
  OUT_even_booking-ids := SYS.ODCI NUMBER LIST();
FOR rec IN(
      SELECT Booking ID
      FROM Booking
      WHERE Temple ID = in_temple_id
      AND MOD (Booking ID, 2)=0
    )LOOP
    out-even_booking_ids(out_even_booking_ids.count):=
            rec.Booking ID;
      END LOOP;
    END;
/
```

Execution Block:

SQL

```
DECLARE
  temple_id NUMBER := 101;
```

```
DBMS_OUTPUT.PUT_LINE('Even Booking ID:' || even_Booking_ids);
    temple_id Number := 101;
even_booking_ids SYS.ODCI Number List;
BEGIN
  GET Even Booking IDs for Temple (temple_id, even_booking_ids);
FOR i IN 1.. even_Booking_ids.COUNT Loop
  DBMS_OUTPUT.PUT_LINE('Even Booking
      ID:' || even_booking_ids(i));
  END loop;
  END;
/
```

VEL TECH - CSE	
EX NO	7
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	4
RECORD (5)	-
TOTAL (20)	14
SIGN WITH DATE	

Result:

Thus, the Triggers, views and exceptions for the Temple ticket online booking management system were successfully implemented and verified.

TASK NO: 8 Date: 11/09/2025 31

CRUD operations in Document Data base.

Aim: To perform Monagoose using Nim 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

Step-2: Install Mongosh

Download MongoDB Shell

Step-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.

Step-4: Confirm PATH by typing in Command prompt:

mongosh --help

Step-5: Open MongoDB Server

C:\program files\mongoDB\Server\bin\mongod.exe

Step-6: perform CRUD Operations.

CRUD operations for Online temple Ticket BMS:

1. Create Collection:

db. createCollection ("Temple Tickets")

{"ok": 1}

Output :-

2) {
"acknowledged": true,
"inserted Id";
objectId("65161172 6ebb te199 3adf 999")
}

3) {
"_id" = objectId("651cf172 6ebbfe999"),
"Ticket ID": "T1001",
"Temple Name": "Meenakshi Amman Temple",
"Location": "Madurai",
"Booking Date",: "2025-09-10",
"Ticket price": 200

4) {
"acknowledge": true,
"inserted Ids": [
objectId("65KF17266be799 3901"),
objectId("65KF1726 ebb te799 3902"),
objectId("651KCF17266ebb te 799 3901"),
]
}

2. Insert
db. templ
Ticket
Temple
Locat
Devote
Boo
Ti
3)
3. Find
db.
4. A M
db.
{
T
Ter
{
5.

2. Insert a single Document:

db.templeTickets.insertOne({

Ticket ID: "T1001",

Temple Name: "Meenakshi Amman Temple",

Location: "Madurai",

Devotee Name: "Ramesh Kumar",

Booking Date: "2025-09-010",

Ticket price: 200.

})

3. Find one Document

db.templeTickets.find({Ticket ID: "T1001"})

4. Insert multiple Documents:

db.templeTickets.insertMany([

{

Ticket ID: "T1002",

Temple Name: "Brihadeeshwara Temple",

Location: "Thanjavur",

Ticketprice: 150

},

{

TicketID: "T1003",

Temple Name: "Kapaleeshwarar Temple",

Location: "Chennai",

Ticket price: 100

},

])

5. Find All Documents

db.templeTickets.find().pretty()

5.
{ "Ticket ID": "T1001",
"Temple Name": "Meenakshi" Aman temple",
"Location": "Madurai",
"Ticket price": 200
}

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

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

6. Upda
d b

7. D

6. Update a Document:

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

7. Delete a Document

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

WEIGHT - 5%	
EX NO	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	—
TOTAL (20)	15
SIGN WITH DATE	T.M. 11.9.25

Result:

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

OUTPUT :

d. devotee ID	d. Name	d. Age	d. phone
2001	Raj Kumar	30	9876543210

OUTPUT :

t. temple ID	t. Name	t. location	t. capacity
1001	Sri venkatesh -wara temple	Tirupati	5000

Task-9 18-09-25 34

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
* copy the clown loaded password file into & login to the Aura console.
* start executing queries in the Neouj Browser.

Create Node with properties:

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

Create a Devotee Node:

CREATE (d: Devotee & Devotee ID: '2001',
Name: 'Raj Kumar.
Age: 30,
phone: '9876543210',
Return d

Create a Temple Node:

CREATE (t: temple & Temple ID: '1001',
Name: 'Sri Venkateshwara Temple.
location: 'Tirupati'.

Output :-

b.BookingID

B001

b.devoteeID

D001

b.TempleID

T001

-> CReate

CREATE

De

Te

Retu

Create

-> Dev

MAT

CRE

RE

-> B

M

C

R

D

Output :-

* Show nodes and relationship [FOR] connecting Booking to temple.

→ Create a Booking Node:

```
CREATE (b: Booking { Booking ID: 'B007',
        Devotee ID: 'D007'
        Temple ID: 'T007',
    Return b.
```

Create Relationships:

→ Devotee makes Booking.

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

→ Booking linked to temple.

```
MATCH (b: Booking { Booking ID: 'B007' })
CREATE (b)-[:FOR]->(t)
RETURN b, t.
```

Display All Nodes:

```
MATCH (n) RETURN n.
```

Retrieve particular Booking details.

```
MATCH (b: Booking { Booking ID: 'B002' })
RETURN b.
```

Update particular Devotee details.

```
MATCH (d: Devotee { Devotee ID: 'D007' })
SET d.phone = '9876543210', d.Age = 31
RETURN d.
```

SELETE Particular Booking

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

DELETE b.

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

18.09.24

Result: By following the above Steps succeesfully created, inserted, queried, update and deleted nodes and relationships in the temple ticket Booking management System using Neow4j Graph Database.

Task-10

25-09-25

Normalizing Databases using functional dependencies upto third Normal form.

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

Given Relation:

TempleBooking (
BookingID, UserID, Uemail, Ucontact, TempleID, TName, TLocation, PriestID, priestName, PriestContact, TicketID, BookingDate, DarshanDate, Timeslot, TicketType, price, PaymentID, PaymentMode, PaymentStatus, TransactionDate
)

Step(a): Apply Functional Dependencies → Normalize to 1NF

First, identify functional dependencies (FDs):

1. UserID → UName, UEmail, Ucontact
2. TempleID → TName, TLocation
3. priestID → priestName, priestContact
4. TicketID → TicketType, Price
5. PaymentID → paymentMode, PaymentStatus, TransactionDate
6. BookingID → UserID, TempleID, PriestID, TicketID, BookingDate, DarshanDate, Timeslot, PaymentID.

Since all attributes already contain atomic values (no repeating groups, no multivalued attributes), the relation is already in 1NF.

Step(b): Normalize using FD+ and R+

Candidate Keys:

→ BookingID (main unique identifier of a booking).

From all (closure of candidate Keys):

- BookingID$^+$ = {All attributes} → confirms BookingID is a candidate key.
- UserID, TempleID, PriestID, TicketID, PaymentID. also act as foreign keys with their own dependencies.

Step(c): Minimal cover / canonical cover

we reduce FDs to minimal form:

1. UserID → UName, UEmail, UContact
2. TempleID → TName, TLocation.
3. priestID → priestName, PriestContact
4. TicketID → TicketType, Price.
5. Payment → PaymentMode, PaymentStatus, TransactionDate.
6. BookingID → UserID, TempleID, priestID, TicketID, BookingDate, DarshanDate, Timeslot, paymentID.

✓Already minimal and canonical → no redundant attributes.

Step(d): Normalize to 2NF.

Conditions for 2NF:

→ Must already be in 1NF.

→ NO partial dependency (non-prime attributes depending only on Part of a composite key.)

Here, BookingID is the primary key (not composite)

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 (non-prime attribute depending on another non-prime).

check:

- BookingID → UserID → UName (transitive dependency).
- BookingID → templeID → TName, TLocation (transitive dependency).
- BookingID → priestID → priestName (transitive dependency)
- BookingID → TicketID → TicketType, Price (transitive dependency).
- BookingID → PaymentID → paymentMode, paymentStatus (transitive dependency).

final simplified tables in 3NF

1. User Table

   User (UserID [PK], UName, UEmail, UContact)

2. Temple Table

   Temple (TempleID [PK], TName, TLocation)

3. priest Table

   priest (priestID [PK], priestName, priestContact)

4. Ticket Table

   Ticket (TicketID [PK], TicketType, Price)

5. Payment Table

   Payment (PaymentID [PK], PaymentMode, PaymentStatus, TransactionDate)

6. Booking Table.

   Booking (BookingID [PK], UserID [FK], TempleID [FK], priestID [FK], TicketID [FK], PaymentID [FK], BookingDate, DarshanDate, Timeslot).

Constraints:

- Primary Keys: BookingID, UserID, TempleID, priestID, TicketID, PaymentID.
- Foreign Keys:

- Booking.UserID → User.UserID
- Booking.TempleID → Temple.TempleID
- Booking.priestID → priest.priestID
- Booking.TicketID → Ticket.TicketID
- Booking.paymentID → payment.paymentID

* Unique Constraints: UEmail, UContact.
* check Constraints:
  - price > 0
  - payment status ∈ {'Pending', 'completed', 'Failed'}
  - timeslot within valid darshan timings.

VEL TECH - CSE	
EX NO	10
PERFORMANCE (5)	4
RESULT AND ANALYSIS (5)	4
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	17
SIGN WITH DATE	25.09.25

Result:

Thus, the given relation for the online Temple ticket Booking management System has been normalized into simplified tables up to Third Normal Form (3NF) with suitable constraints, reducing redundancy & improving data integrity.

TASK-11

Aim :- To design an online temple booking system management using oracle forms, Menus & report builder. The system involves creating a user interface for booking ticket online, managing bookings and generating reports related to tickets 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 data blocks that represents key tables a view such as

* Temples
* ticket types

* Booking details
* users/devotes
* payment information.

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

Create Menus:

Menus provide the navigation for the booking system application.

Steps to create Menus in oracle Forms:-

1. open oracle forms builder
2. create a new menu form a us an existing one
3. Add menu for each major function such as:
   - Book Tickets
   - View Booking History
   - Canul Tickets
   - Manage Temples
   - Reports
4. Define Menu hierarchy
5. Assign triggers or produces to handle menu item action.
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 canullation form.
2. Add form elements like text fields buttons, and list.
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
   - validality booking date
   - checking booking dates
   - processing payments
5. test the forms inside oracle forms builder for functionality and data accuracy.

Create Reports:

Reports provide summarized information about booking and temple visits

1. open oracle Report Builder
2. create a new report or use an existing template.
3. Define the data source for the report using queries or PL/SQL procedures.

Example reports:

Daily Ticket Sales report
Booking summary by Temple
Revenue Report.

4. Design the report layout with headers
5. Add parameter to allow filtering
6. Generate and preview the report to verify the data accuracy and presentation.

VEL TECH - CSE	
EX NO	1.
PERFORMANCE (5)	✓
RESULT AND ANALYSIS (5)	✓
VIVA VOCE (5)	✓
RECORD (5)	✓
TOTAL (20)	20
SIGN WITH DATE	29.10.24

Result:

Thus, designing an online temple ticket booking management system with oracle forms, menus and report builder involves creating an interactive UI for ticket booking and managing operations, along with generating detailed reports. The system is successfully developed to facilitate online bookings and efficient temple management.

16/10/25 10SK-12

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

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)
```

16/10/25

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