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

211CA207 – DATABASE MANAGEMENT SYSTEMS

NAME: K. Madhu Sumanth

REG.NO: VTU 27547

ROLL.NO: 24UECL0025

BRANCH: CSE(AI&ML)

YEAR/SEM: 2nd year / 3rd Sem

SECT: S10L12

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

AME	: K. Madhu Sumanth	BRANCH	: CSE(AI&ML)
TU NO.	: 27547	REG.NO.	: 24UECL0025
EAR/SEM	: 2nd year / 3rd sem	SLOT NO.	: S10L12

ertified that this is a bonafide record of work done by above udent in the **"10211CA207-DATABASE MANAGEMENT STEMS LABORATORY"** during the year 2025-2026 (Summer mester).

31/10/25

GNATURE OF LAB HANDLING FACULTY SIGNATURE OF HOD

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

ERNAL EXAMINER EXTERNAL EXAMINER

Scanned with OKEN Scanner

INDEX

Vecra's Student's Choice

Name : K. Madhu Sumanth

STD : 2nd yr SEC : CC(AI&ML) ROLL NO : 5

S.No.	Date	Title	Page No.	Teacher's Sign /

INDEX

	Date	Title	Page No.	Marks	Faculty Signature
	24/07/25	Conceptual Design after Formal Technical Review	1	17	24/7
	31/07/25	Generating design of other traditional database models	6	18	31/7
	07/08/25	Developing queries with DML Single-row functions and operations	16	17	7/8
	14/08/25	Developing queries with DML Multi-row functions and operations	21	19	14/8
	21/08/25	Writing Join Queries, equivalent, and/or recursive queries	26	17	21/8
	28/08/25	Writing PL/SQL using Procedures, Function	30	17	28/8
	04/09/25	Writing PL/SQL using Loops	39	20	4/9
	11/09/25	Normalizing databases using functional dependencies up to BCNF	43	19	11/9
	18/09/25	Backing up and recovery in databases	46	18	18/9
	25/09/25	CRUD operations in Document databases	50	18	25/9
	09/10/25	CRUD operations in Graph databases	54	18	9/10
	16/10/25	Micro Project: Gift Coupon Application	58	18	16/10

Completed

Total Marks: 216/240

R.T. Thyagu

Signature of Faculty

Scanned with OKEN Scanner

Date :- 24/07/25

Task - 1

A. Temple ticket Online Booking Managment System

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

Entity :

The real-world object or concept that can be distinctly identical. Examples included students, Courses, or products.

Set :-

A collection of Entities of the same types. 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.

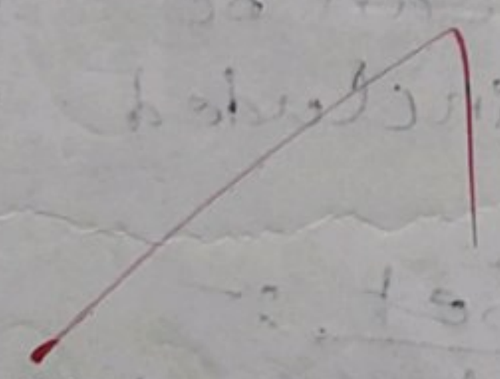
TASK 1 : Conceptual Design after

FTR

(~~Formal Technical Review~~)
(Full Text Retrieval)

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

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



Relationship :-

An Association or interaction between two or more Entities, for Example, a student might Enroll in a Course. Establishing a relation between the "Student" and "Course" Entities,

Aim :

To design and develop an Entity-relationship (ER) Model for a Temple ticket online Booking Management System that allows devotees to book tickets for temple visits, Special darshans, Poojas queues and improving crowd management.

Attributes :-

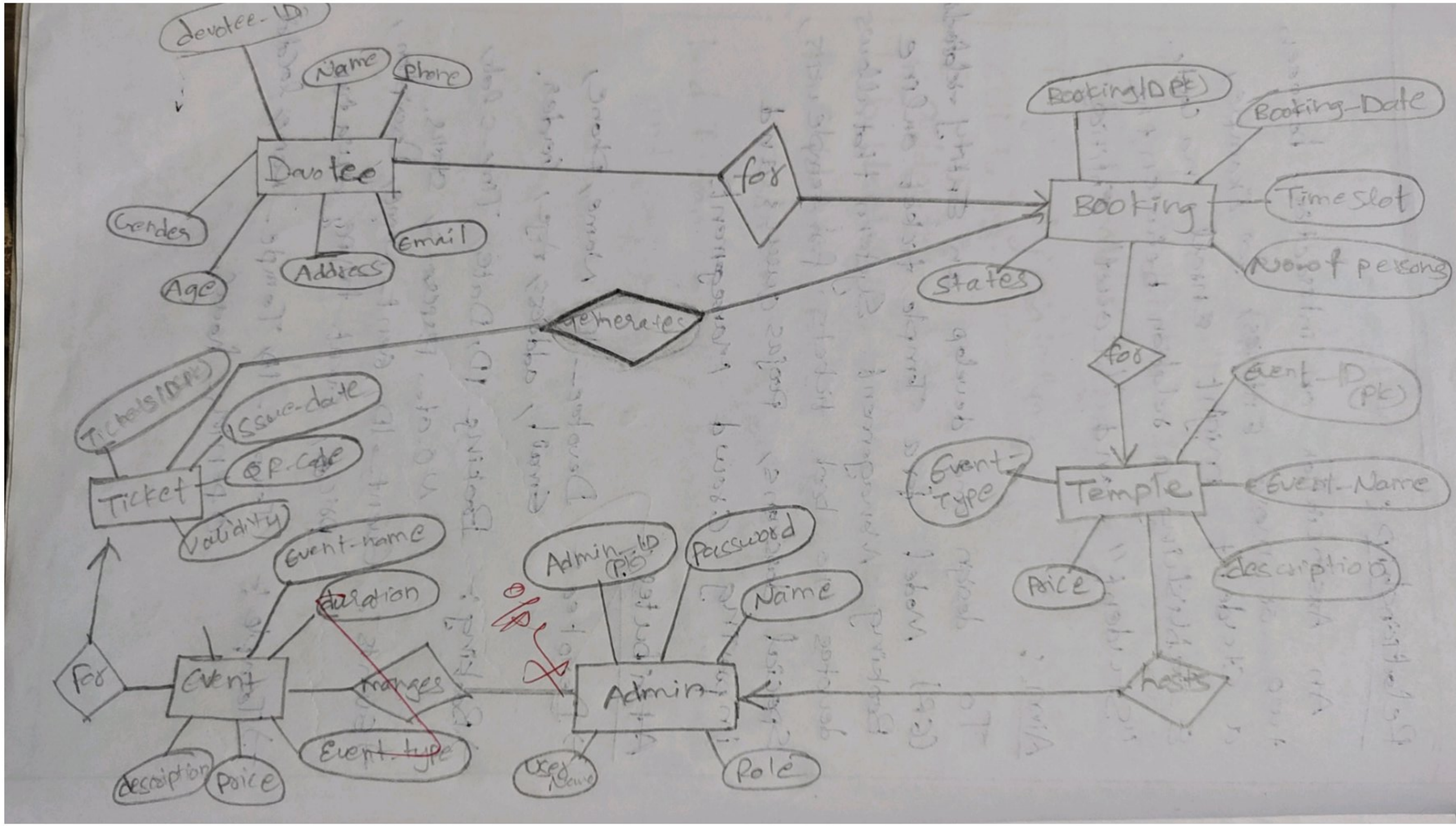
→ Devotee :- Devotee_ID, Name, Phone, Email, address, Age, Gender.

→ Booking :- Booking ID, Date, Time-slot, No.of_persons, status.

→ Events :- Event_ID, Event_Name, Description, price, Event_type, duration.

→ Temple :- Temple_ID, Temple_Name, location, Opening_hours.

Scanned with OKEN Scanner


devotee-ID
Name
Phone
Devotee
Gender
Age
Address
Email
for
Booking ID (PK)
Booking-Date
Booking
Time slot
No of persons
States
generates
Tickets ID (PK)
Issue-date
QR-Code
Ticket
Validity
for
Event-ID (PK)
Event-Type
Temple
Event-Name
Price
description
hosts
Event-name
duration
For
Event
manages
Admin-ID (PK)
Password
Name
Admin
Event-type
description
Price
User Name
Role

→ Payment : Payment-ID, Amount, Method, Status, Date

→ Ticket : Ticket-ID, issue-Date, QR-Code, Validity

→ admin : Admin-ID, Name, Username, Password, Role.

Types of Attributes :-

→ Simple : Name, Price
→ Composite : Address (street, City State).
→ Multi-valued : Phone
→ Derived : Age (from DOB)

Relationships :-

- Devotee → Booking (makes)
- Booking → Event (for)
- Booking → payment (has)
- Booking → Ticket (generates)
- Events → Temple (hosts)
- Admin → event (manages)

Scanned with OKEN Scanner

- Relationship Types

→ One - to - one : Booking → Payment, Booking → Ticket

→ 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.

VEL TECH - CSE

EX NO	1
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	2
RECORD (5)	5
TOTAL (20)	2+5=17
SIGN WITH DATE	

4/4/25

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: 31/7/25

Task-2

Generating Design of other traditional database model.

Aim :-

cocating Hierarchial / Network model of the database 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 heirarchy / has-a hierarchy and performs generalization and /or specialization relationship.

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

2.d Rename the relations.

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

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

~~SP~~
Relationship 1: Temple manages Event (One-to-many)

specifity: 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-ma

→ Specificity: Each event can have many Tickets types, but each Ticke type belongs to one Event.

→ Surplus Relation: No surplus relation needed; one-to-many is appropriat

Relationship 3: Customer books Ticket (many-to-many

→ Specificity: A customer can book many Tick and a Ticket can be booked by many Customers.

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

Relationship 4: Ticket allocated to Seat (one-to-one)

→ Specificity: Each ticket corresponds to a specific seat, and each seat can be assigned to only one Ticket.

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

Scanned with OKEN Scanner

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

Generalization Example:

→ Entities: Customer, Admin

→ Common attributes: User_ID, First_Name, Last_Name, Email, Contact_Number, password.

→ Generalized superclass: User with above attributes

→ Subclasses:

→ Customer: Additional attributes like Customer_ID, Address, Payment_info

→ Admin: Additional attributes like Admin_ID, Role.

Specialization Example:

→ Entity: Ticket

→ Specialized into:

→ General Ticket (attributes: price, general_access_area)

→ VIP Ticket (attributes: price, access_to_special_area, complimentary_seating)

Scanned with OKEN Scanner

2.c. Find the domain of attributes and perform check constraints.

Attribute	Domain	Check Constraint Example
age (customer)	Positive integer (minimum 18)	CHECK (age >= 18)
ticket_price	positive decimal number	CHECK (ticket_price > 0)
booking_date	Date (not in past)	CHECK (booking_date >= CURRENT DATE)
seat_number	String or number within valid range	CHECK (seat_number BETWEEN 1 AND 500)

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

ALTER TABLE Customer RENAME COLUMN Contact_no To phone_

ALTER TABLE Ticket RENAME COLUMN price To ticket_price;

ALTER TABLE Booking RENAME COLUMN book_date To booking_date

Scanned with OKEN Scanner

DCL (Data Control Language)

CREATE USER temple_user IDENTIFIED BY password 123;

GRANT CONNECT, RESOURCE TO temple_user;

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

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

GRANT SELECT, INSERT, UPDATE, DELETE ON Ticket to temple_user;

GRANT SELECT, INSERT, UPDATE, DELETE ON event to temple_user;

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

Scanned with OKEN Scanner

2.c. Perform SQL Relations using DDL and DCL commands.

Data Definition Language (DDL):

- Create User Table (Generalization)

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

- Specialized Customer Table

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

User

User ID (PK)
First Name
Last Name
Age
Date of Birth
Email
Contact_NO
Role

Customer

Customer ID(PK)
Address
Payment Info

Admin

Admin ID(PK)
Role Description

Scanned with OKEN Scanner

- Admin Table

```
CREATE TABLE Admin (
    Admin_ID INT PRIMARY KEY,
    User_ID INT,
    Role VARCHAR(50),
    FOREIGN KEY (User_ID)
  REFERENCES User (User_ID)
);
```

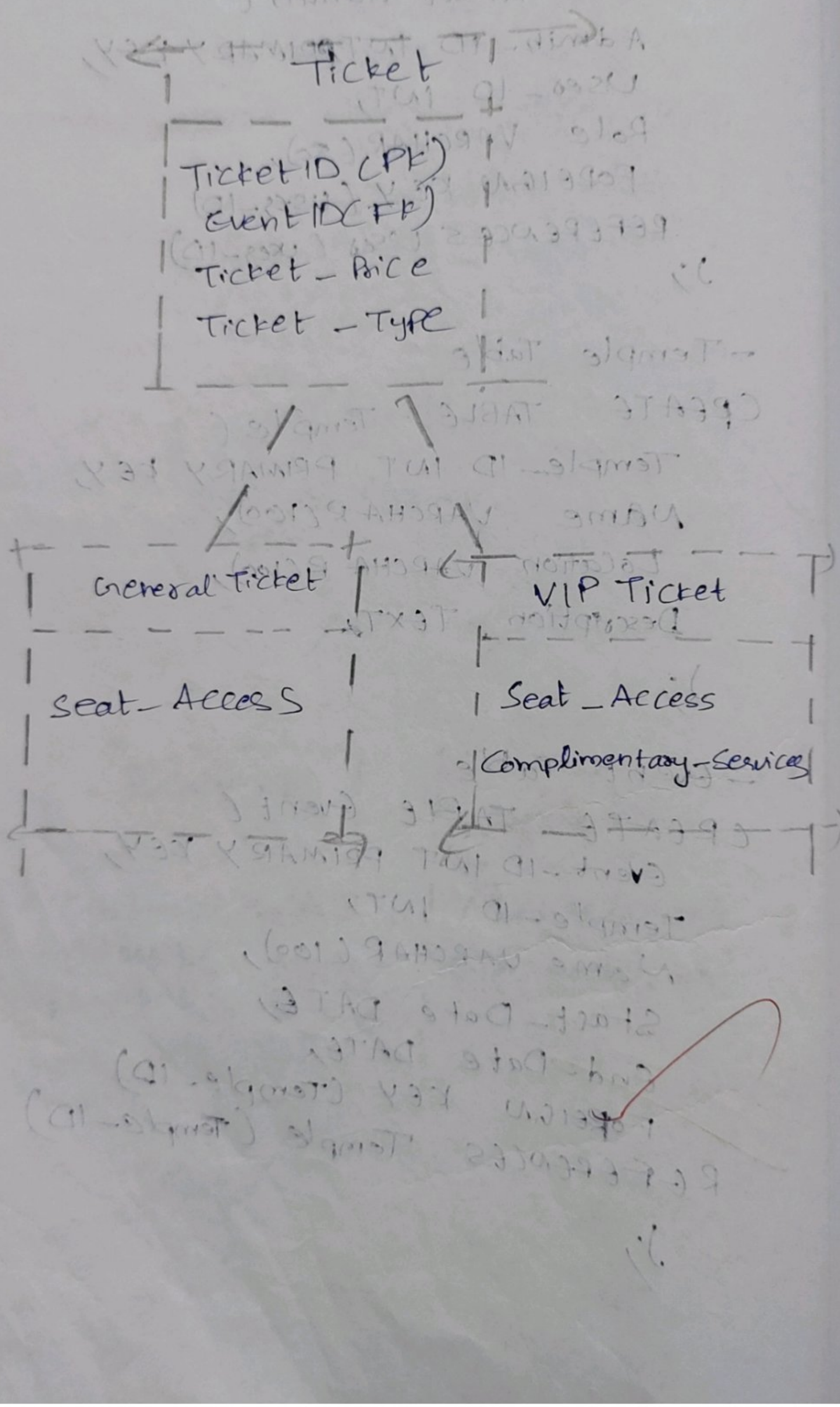
- Temple Table

```
CREATE TABLE Temple (
    Temple_ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Location VARCHAR(100),
    Description TEXT
);
```

- 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)
);
```

Scanned with OKEN Scanner


Ticket
Ticket ID (PK)
Event ID (FK)
Ticket_Price
Ticket_Type
General Ticket
VIP Ticket
Seat_Access
Seat_Access
Complimentary_Services

- Ticket Table with Specialization handled by ticket_type attribute

```
CREATE TABLE Ticket(
    Ticket_ID INT, PRIMARY KEY,
    event_ID INT,
    Ticket_Type VARCHAR(20),
    'General', 'VIP'
    Ticket_price DECIMAL(10,2),
    FORGIGN KEY (Event_ID)
  REFERENCES Event (Event_ID)
);
```

-- Booking Table (surplus relation for many-to-many)

```
CREATE TABLE Booking(
    Booking_ID INT PRIMARY KEY,
    Customer_ID INT,
    Ticket_ID INT,
    Booking_Date DATE,
    Seat_Number INT,
    FOREIGN KEY
  (Customer_ID) REFERENCES
  Customer (Customer_ID),
    FORGIGN KEY (Ticket_ID)
  REFERENCES Ticket (Ticket_ID)
);
```

Scanned with OKEN Scanner

VEL TECH - CSE

EX NO	2
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	13+5=18
SIGN WITH DATE	

3/10/25

Result: Thus, the Hierarchical and Network models have been successfully created for the Temple Ticket Online Booking Management System, with appropriate inheritance, relationship specificity, domain constraints, renamed relations, and SQL implementations using DDL and DCL commands.

Date: 7/8/25

Task-3

Using clauses, operators and functions in queries!

Aim:- To perform query processing on a Temple ticket online Booking Managment system for different retrieval results of queries using DML, DDL operations with aggregate functions, data function, string function, set clauses, and operators.

Temple:

TEMPLE ID	NAME	LOCATION	CONTACT_No
TID01	Tirumala Venkatesh wave	Tirupati	9024276954
TID02	meenakshi Amman	Madurai	7989222725
TID03	bashi vishwanath	Varanasi	7013982716
TID04	Jagannath Temple	Puri	9701224404
TID05	Golden Temple	Amritsar	9908616499

Scanned with OKEN Scanner

Visitor

Visitor In	F Name	L.Name	age	Email	Contact.No
V001	Anil	Kumar	30	Anilk@gmail.com	9014276954
V002	Akash	Reddy	25	Akash@gmail.com	970122 4404
V003	Priya	Sharma	28	Priyas@gmail.com	99082 90010
V004	Aarav	Patel	35	Aaravp@gmail.com	70139 82716
V005	Sneha	Rao	22	Snehas@gmail.com	798922 2725.

Booking

Booking ID	Temple ID	visitor ID	Booking-date	Ticket Type	amount
B001	TID01	V001	2024-06.15	VIP	500
B002	TID02	V002	2024-06-16	General	100
o/p B003	TID01	V003	2024-06-17	General	100
B004	TID03	V004	2024-68	VIP	700
B005	TID03	V005	2024-06-19	General	200

Priest

Priest ID	FName	CName	Age	Email	Contact-No
P001	Ramesh	Iyer	50	ramesh@gmail.com	9014076841
P002	Suresh	Sharma	45	Suresh@gmail.com	9701224404
P003	Manish	Das	40	manish@gmail.com	7789222724

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

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

Result:

Visitor ID	FNAME	CName	Age	Contact-No
V002	Akash	Reddy	25	9701224404
V004	Aarav	Patel	35	7089782716

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~~ Booking
WHERE Ticket_Type = 'VIP';

Result :-

Count(*)
2.

5. Display temple details for temple IDS 'TIDO1', 'TIDO3', and 'TIDO4'.

SELECT *
FROM TEMPLE
WHERE TEMPLE IN ('TIDO1', 'TIDO3', 'TIDO4');

Result :

Temple ID	Name	location	Contact-No
TIDO1	Tirumala	Tirupati	9701224404
TIDO2	kashi	Varanasi	7013952746
TIDO3	Golden-temple	Amritsar	7982222725.

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

Select visitor ID, FNAME, CNAME
FROM visitor
WHERE visitor ID IN (
SELECT visitor ID FROM Booking WHERE
Ticket_type = 'Special'
);

Result :

Visitor ID	FName	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 :

priest ID

(No Result if all priests are assigned)

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

7/8

Result : Thus, query processing for temple ticket online Booking Management System using clauses, operators, and functions was successfully performed.

Date :- 14/08/25

Task - 4

Using function in Queries and writing Sub Queries :-

Aim :- To perform advanced query processing and test its heuristic by designing optimal correlated and nested subqueries such as finding summery statics in the temple ticket online Booking Managment system.

4.1 : To retrieve all temple details, including the count of booking 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.
LEFT JOIN Booking b
ON t. Temple ID = b. Temple ID
Group By t. Temple ID, t. Name,
t. location, t. Contact_No;
```

Output:

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

4.2: To retrieve the Total number of 'Special' seve 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 Venkatsuma	1

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

```
SELECT *
FROM Temple
WHERE Temple ID IN C
```

SELECT Temple ID
FROM Booking
WHERE Ticket_Type = 'VIP'.
);

Output:

Temple ID	Name	location	Contact-No
T1D01	Tirumala	Tirupati	9760246041
T1D03	Kashi	Varanasi	7013982716

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

SELECT v. visitor ID.
v. FName As visitor Name,
v.age,
b. Booking ID,
b. Booking_Date,
b. Ticket_Type,
b. amount.
FROM visitor V
ID IN Booking b
ON v. visitor ID = b.visitor ID
WHERE v. Age > 25;

Output:

Visitor ID	visitor Name	Age	BookingID	Amount
V001	Anil	30	B001	500
V003	Priya	28	B003	100
V004	Aarav	35	B004	700

Scanned with OKEN Scanner

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

```
SELECT*
FROM TEMPLE
WHERE TempleID NOTIN (
 SELECT TempleID
 FROM Booking
 );
```

Output :

Temple ID	Name	location	Contact_No
TID04	Jagannath Temple	Puri	9701224404
TID05	Golden temple	Amritsar	9014276954

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

```
SELECT t.TempleID,
       t.Name As Temple Name,
FROM Temple t.
JOIN Booking b
ON t.TempleID = b.TempleID
JOIN Visitor V
ON b.visitorID= v.visitorID
WHERE V.VisitorID = 'V005';
```

Output :

Temple ID	TempleName	VisitorName
TID01	Tirumala	Srcha

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

14/8

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

Temple	Pooja Name	Price
Tirupati Temple	Suprabhata Seva	300
Tirupati Temple	Archana	100
Kanchi Temple	Abhishekam	500
Madurai Temple	Special Darshan	250

Output:

Booking ID	Devotee	Pooja Name	Bookingdate	Slottime	Status
B101	Rajesh	Suprabhata Seva	22-jun-23	06:00AM	Confirmed
B102	Meena	Archana	22-jun-23	9:00AM	Confirmed
B103	Arjun	Abhishekam	23-jun-23	7:30AM	Cancelled
B104	Kavitha	Special Darshan	23-jun-23	8:30AM	Confirmed
B105	Suresh	Abhishekam	24-jun-23	6:30AM	Confirmed

Output

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

21/8/25

TASK 5

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

Aim: To perform advanced query processing and Test its heuristics using join queries, equivalent queries, and recursive queries for The Temple Ticket Online Booking Management System, which manages devotees, priests, poojas, tickets, and booking details.

Queries

5.1 To retrieve all temples and their poojas

sql

```
SELECT t.Name as Temple, P.PoojaName
  P.Price FROM Temple t
JOIN Pooja ON t.TempleID=P.Temple
                              ID,
```

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.Slot time,
  b.Status FROM Booking b
  JOIN Devotee N b.DevoteeID
                    =d.DevoteeID
JOIN Pooja P ON b.PoojaID =P.Pooja
                              ID.
```

Output :

Devotee ID	FName	Mobile No
D 103	Arjun	9876543210
D 105	Suresh	8765432190

Output :

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

Output

Temple	Cancelled Bookings
Kanchi Temple	1

Output :

TempleID	Name	Location	Head Priest
T001	Tirupati Temple	Tirupati	Ramanujam
T002	Kanchi Temple	Kanchipuram	Narayan Das
T003	Madurai Temple	Madurai	Subramanyam

Scanned with OKEN Scanner

5.3 Count the number of bookings made by each devotee

Sql

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

5.4 To find all devotees who booked 'Abhishekam'

Sql

```
SELECT d.Devotee ID, d.FName, d.Mobile
 No
FROM Devotee d
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 retrieve all Poojas and the number of times they were booked

Sql

```
SELECT P.Pooja Name, COUNT (b.Booking
 ID)
 AS Total Bookings
FROM Pooja P
LEFT JOIN Booking b ON P.Pooja
 ID = b.Pooja ID
~~GROUP~~ GROUP By Pooja Name.
```

Output :

FName	Age	Slot Time	Booking ID	Booking date	Status
Suresh	55	6:30AM	B105	24-jun-23	Confirmed

Output :

TempleID	Name	Location	HeadPriest
T004	Ramashwaram Temple	Rameshwaram	Vishnu Devan

Output :

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

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

Sql

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

5.7 To retrive temple details where booking were successfull

Sql

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

5.8 To retrive devotee and booking details from devotees above 50 years old.

sql

```
SELECT d.FName, d.Age, b.BookingID,
b.Booking Date, B.Slot time, b.status
FROM Devotee d
JOIN Booking b ON d.DevoteeID=b.Devotee
                ID
WHERE d.Age > 50;
```

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

Sql

```
SELECT *
FROM Temple
WHERE Temple ID NOT IN
        (SELECT Temple ID
         From Booking.
```

~~5.10~~

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

Sql

```
SELECT t.Name As temple, P.PoojaName,
  d.F.Name AS Devotee, b.BookingDate,
  b. Slot time
FROM Booking b
JOIN Temple t on b.Temple ID = t.TempleID
JOIN Pooja p on b.PoojaID = P.PoojaID
JOIN Devotee d on b.DevoteeID =
                d.DevoteeID
where b. BookingID = 1;
```

EX NO	5
PERFORMANCE (5)	4
RESULT AND ANALYSIS (5)	4
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	17
SIGN WITH DATE	1.10.24

Result: Thus, the temple Ticket Online Booking management system queries using Queries equivalent queries and recursive queries were executed successfully to manage temples, devotees, priests, pujas and bookings.

Date: 28/8/25

Task-6

Procedures, functions, and loop using PSL/SQL on Number theory and Business scenarios.

Aim: To write and execute PSL/SQL procedures, functions and loops on Number theory (eg, prime number, factorial) and business scenarios (eg Salary Calculation, Discount Calculation).

Software used: Oracle SQL/PL/SQL

Algorithm:-

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

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

(B) Business scenario Example - Salary Bonus calculation using function

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

Programs :-

A) Number Theory - Prime Number Check
(Procedure with Loop)

```
SET SERVER OUTPUT ON;
CREATE OR REPLACE PROCEDURE
check_prime (n IN Number) is
    flag BOOLEAN := TRUE;
BEGIN
 If n<=1 THEN
    DBMS_OUTPUT.PUT.Line
        (n||' is NOT prime');

ELSE
 FOR i in 2..n/2 loop
    if MOD(n,i)=0 then
        flag := FALSE;
        EXIT;
    ENDIF;
    END LOOP;
    IF flag THEN
        DBMS_OUTPUT.PUT_LINE
            (n||' is prime');
    ELSE
    DBMS_OUTPUT.PUT_LINE
            (n||' is NOT Prime');
    END IF;
    ENDIF;
END;
/
Execution
BEGIN
```

```
    check_prime(29);
    check_prime(20);
END;
/
```

(B) Business 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;
  ELSEIF salary BETWEEN 20000
                AND 50000 THEN
      bonus := Salary * 0.05;
  ENDIF;
  RETURN (salary + bonus);
END;
/
-- execution
DECLARE
  final_sal NUMBER;
BEGIN
  final_sal := calc bonus (18000);
DBMS_OUTPUT.PUT_LINE
      ("Final salary with Bonus:"
          || final_sal);
```

Output:

For Prime Number Check:

29 is prime

20 is NOT prime

FOR Bonus Calculation :

Final Salary with Bonus : 21 600

Final Salary with Bonus : 33 000

```
final_sal := Calc_bonus(30000);
DBMS_OUTPUT.PUT_LINE
        ("Final Salary with Bonus:"
            || final_sal);

END;
/
```

VEL TECH - CSE	
EX NO	6
PERFORMANCE (5)	4
RESULT AND ANALYSIS (5)	4
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	20 = 17
SIGN WITH DATE	04.09.25

Result: Thus, PL/SQL procedures functions, and Loops were successfully implemented for Number Theory and Business Scenario.

Scanned with OKEN Scanner

Date: 4/9/25

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

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

Sql

```
CREATE OR REPLACE TRIGGER
trg_insert_ticket
AFTER INSERT ON BOOKING
FOR EACH ROW
BEGIN
  INSERT INTO Ticket (TicketID,
                      BookingID,
  DevoteeName, Age, Gender, QRcode,
  Status)
  VALUE SC
    seq_ticket.NEXT VAL, -- assuming a
  sequence for ticket IDs
  : NEW.Booking ID,
  'Unknown'... Default placeholder
  'NULL' ... age not given at booking time
  'NULL', -- gender not given at booking time.
```

SYS.GUIID()... auto-generate QR code.

'ACTIVE'... default ticket status.

);

END;

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 Darshan Type,
    s.StartTS As Slotstart,
    s.EndTs 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 slots ON b.Slot ID = S.Slot ID
```

Scanned with OKEN Scanner

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, EXTEND;
Out_even_booking_ids (out_even_
                        booking_ids
                        COUNT): =
                        rec. Booking ID;
    END LOOP;
    END;
/
```

Execution Block:

Sal

```
DECLARE
  temple_id NUMBER:= 101;
    even_booking_ids. SYS.ODCI NUMBER
                        LIST;
  BEGIN
    Get Even. Booking IDs For Temple
                (temple_id, even_booking
                ids);

    FOR i IN 1.. even_ids. COUNT LOOP
      DBMS_OUTPUT. INE ('Even Booking
                        ID:'
                        ||even_Booking_ids (i));
      temple_id. Number:= 101;
      even_booking_ids SYS.ODCI
                        Number List:
```

o/p

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

VEL TECH - CSE	
EX NO	7
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	25 = 20
SIGN WITH DATE	T.M.Thr 04.09.25

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

Scanned with OKEN Scanner

Date: 11/9/25

Task - 8

AIM: To design a document database from online Temple ticket booking management system using Mongoose (NoSQL) with Mongo DB and Perform CRUD operations Create, read, update, Delete tickets, temples, and user bookings.

Algorithm:

Step 1: Install mango DB Community Editions.

Step 2: Install Mango shell (Mangosh)

Step 3: Configure PATH Environment variable to include Mangosh binary

Step 4: Confirm installation
Bash
mongosh --help

Step 5: Start mongo DB Server
Bash
C:\Program Files\Mongo DB\server\bin\mongod.exe

Step 6: Perform CRUD operations on Temple Ticket booking Database

OUTPUT:

{
"acknowledged": True
"insested ID":
object ID" 65ldlcaaebbfe799 3adfe"
}

OUTPUT:-

{
"aknowledged": true
"insested IDS": C
Object ID C "65ldlcc36ebbfe799 3adf9"
Object ID ("65ldlcc36ebbfe799 3adf98"),
]
}

CRUD operations :

1) i) insert Temples:-

db. Temples. insert one ({
Temple ID : "TID01"
Name : "meenakshia Amman Temple",
Location : "Madurai",
Contact : 9876543210
Timings : "6:00AM – 9:00PM"
});

ii) db. Temples. insert many ([
{ Temple ID : "TID02", Name :
"Ramanatha Swamy Temple", Location :
"Rameswaram", Contact : 9765432109,
Timings : "5:00 AM : 8:00 AM" },
{ Temple ID : "TID03", Name"
"Brihadeswar Temple", Location",
"Thanjavur" Contact : 9654321098,
"7:00AM – 8:30 PM" }
]);

OUTPUT:-

{
"acknowledged" true
"inserted ID"
object "ID"("651dlceobebbfe799 3adfa15")
}

OUTPUT:-

{ "_id":
object ID("651dlc9ebbfe799 3adfa6")
"Temple ID": "TD01",
"Name": "meenakshi Amman Temple"
"Location": "Madurai",
"Contact": 9876543210,
"Timings": "6:00AM - 9:00PM"
}
}

"_id"
object_id("651dcc3be bbfe799 3adf9")
"Temple ID": "TID02",
"Name": "Ramanathaswamy Temple".
"Location": "Rameswaram",
"Contact": 9765432109,
"Timings": 5:00 AM - 8:00 PM
}

2) Insert ticket Booking

```
db.Tickets insert one({
    Ticket ID: "TK1D01",
    Temple ID: "T1D01",
    User Name: "John Doe",
    Booking Date: "2025-09-10",
    Number of persons: 3
});
```

3) Query All Temples

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

"_id":
object ID("65d1cc3bebbfe7993adf112"},
"Temple ID: "TID03",
"Name": "Brihadeshwara Temple",
"Location": "Thanjavur"
"Contact": 9650321698,
"Timings": "7:00AM – 8:30PM"
}

OUTPUT:

{
"acknowledged": true,
"match Count": 1,
"modified Count": 1
}

OUTPUT:-

{
"acknowledged": true,
"deleted Count": 1
}

OUTPUT:-

{
"acknowledged": true
"deleted Count": 1
}

4) Query ticket Booking by username

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

5) Update Temple Contact

```
db.Temples.updateOne(
    {TempleID:"TID01"}
    {set:{contact:9998877776}}
);
```

6) Delete a Ticket Booking

```
db.Tickets.deleteOne({TicketID:"TkID01"});
```

7) Delete a Temple

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

VEL TECH - CSE

EX NO	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	19
SIGN WITH DATE	11.09.24

Result:

Thus using MongoDB with mongoos, we successfully designed a document database for an outline Temple Ticket Booking Management System and performed CRUD operations.

output

d.devotee ID	d.Name	d.Age	d.Phone
D001	Rajkumar	30	9876543210

output

t.TempleID	t.Name	t.location	t.capacity
T007	Sri Venkateswara Temple	Tirupati	5000.

18/9/25

Task-9

CRUD operations in Graph databases:-

Aim :-

To perform CRUD operations like creating inserting, querying, finding, deleting operations on graph spaces. The steps to get started with Neo4j's Aura Graph database.

Step-1 :- Set up Neo4j Aura DB:

1. Open
2. Click Start free → Continue with Google → open.
3. Copy the downloaded password ~~file~~ file into and log in to the Aura Console.
4. Start executing queries in the Neo4j Browser.

Create Nodes with properties:

Each node represents an entity like Devotee, temple and Booking.

Create a Devotee Node :-

```
CREATE (d:Devotee {DevoteeID:'D001',
        Name: 'Rajkumar',
    Age : 30, Phone: '9876543210', Email:
            'rajkumar@gmail.com'})
~~Return~~ RETURN d.
```

Create a Temple Node :-

```
CREATE (t:Temple {TempleID:'T001',
        Name: 'Sri Venkateswara Temple',
    location:'Tirupati', capacity:5000})
Return t.
```

b.Booking ID	d.devotee ID	b.TempleID
B001	D001	T001

Output

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

Create a Booking Node

CREATE (b:Booking {Booking ID:
Temple ID: 'T001', Date: '2025-09-10',
No. of Tickets: 3})

RETURN d

Create Relationships

Devotee makes Booking

MATCH (d:Devotee {Devotee ID: 'D001'}),
(b: Booking {Booking ID: 'B001'})
CREATE (d) -[:MADE]→(b)
RETURN d, b

Booking linked to temple

MATCH (b: Booking {Booking ID: 'B001'}),
(t: Temple {Temple ID: 'T001'}
CREATE (b):[:FOR]→(t)
RETURN b. t

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 {Devotee ID: 'D001'})
SET d.Phone = '9123456780', d.Age = 3,
RETURN d.

Scanned with OKEN Scanner

6: 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	

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

Date: 25/9/25

Task - 10

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:

Temple Booking (
Booking ID, User ID, UName, UEmail, UContact, Temple ID, TName, TLocation, Priest ID, Priest Name, Priest Contact, Ticket ID, Booking Date, Darshan Date, TimeSlot, Ticket Type, Price, Payment ID, Payment Mode, Payment Status, TransactionDate
)

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

First, identify function dependencies (FDS):

1. User ID → UName, UEmail, UContact
2. TempleID → TName, TLocation
3. Priest ID → Priest Name, Priest Contact
4. Ticket ID → Ticket Type, Price
5. Payment ID → Payment Mode, Payment Status, Transaction Date
6. Booking ID → User ID, Temple ID, Priest ID, Ticket ID, Booking Date, Darshan Date, Timeslot, Payment ID ~~Date~~.

INF output Table :-

Customer

Name	product id	Quantity
Alice	P1	2
Alice	P2	3
Bob	P3	7

Step (b): Normalize using FD+ and α+

Candidate keys:

→ BookingID (main unique identifier of a booking).

From α+ (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. PaymentID → PaymentMode, PaymentStatus, TransactionDate
6. BookingID → UserID, TempleID, PriestID, TicketID, BookingDate, DarshanDate, Timeslot, PaymentID.

Step(d): Normalize to 2NF

Conditions for 2NF:

→ Must already be in 1NF.

→ No partial dependency (non-prime attributes)

Here, BookingID is the primary key.

The schema is in 2NF.

Scanned with OKEN Scanner

Output

Min = d

A → B,
B → C,
AB → D
D → A.
}

Output

Student id	Student Name
101	Alice
102	Bob

Scanned with OKEN Scanner

Step (e): Normalize to 3NF

Conditions for 3NF:

→ must already be in 2NF.

→ No transitive dependencies.

Check:

→ Booking ID → User ID → UName

→ Booking ID → Temple ID → TName, TLocation

→ Booking ID → Priest ID → Priest Name

→ Booking ID → Ticket ID → Ticket Type, Price

→ Booking ID → Payment ID → Payment Mode, Payment Status

1. User Table

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

2. Temple Table

Temple (Temple ID [PK], TName, TLocation)

3. Priest Table

Priest (Priest ID [PK], Priest Name, Priest Contact)

4. Ticket Table

Ticket (Ticket ID [PK], Ticket Type, Price)

5. Payment Table

Payment (Payment ID [PK], Payment Mode, Payment Status, Transaction Date)

6. Booking Table

Booking (Booking ID [PK], User ID [FK], Temple ID [FK], Payment ID [FK], Booking Date, Darshan Date, Timeslot)

Scanned with OKEN Scanner

output

Student id (PK)

Student Name

Student ID	Student Name
101	Alice
102	Bob

Output :

1. Student

Student ID	Student Name
101	Alice
102	Bob

2. Course

Course ID	Course Name
C01	Maths
C02	Physics

Scanned with OKEN Scanner

Constraints:

→ Primary keys: Booking ID, User ID, Temple ID, Priest ID, Ticket ID, Payment ID.

→ Foreign keys:

* Booking.UserID → User.UserID
* Booking.TempleID → Temple.TempleID
* Booking.PriestID → Priect.PriestID
* Booking.TicketID → Ticket.TicketID
* Booking.PaymentID → Payment.PaymentID

→ Unique Constraints: UEmail, UContact

→ Check Constraints:

→ Price > 0

→ PaymentStatus ∈ {'Pending', 'Completed', 'Failed'}

→ TimeSlot within Valid darshan timings.

VEL TECH - CSE

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

25/9

Result:

Thus, the given relation for the OnlineTemple Ticket Booking management System has been normalized into simplified tables upto Third Normal Form (3NF) with suitable constraints, reducing redundancy and improving data integrity.

Scanned with OKEN Scanner

Date: 09/10/25

Task 11

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 data blocks that represent key table or views such as:

- → Temples
- → Ticket Types
- → Booking Details
- → Users/Devotees
- → Payment Information

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

Create Menus:

Menus provide the navigation structure for the booking system application.

Steps to create menus in Oracle Forms:

1. Open Oracle Forms Builder.
2. Create a new menu form or use an existing one.
3. Add menu items for each major function, such as:
   - → Book Tickets
   - → View Booking History
   - → Cancel Tickets
   - → Manage Temples
   - → Reports.
4. Define menu hierarchy
5. (eg: Book Tickets → Select Temple → Choose Ticket Type).
5. Assign triggers or procedures to handles menu item actions.
6. Compile and run the menu form ~~the~~ 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 list.
3. Use the property Palette to configure element properties and link data blocks to database elements.
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 summarized information about bookings and temple visits.

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.
   Example reports:
   → Daily 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)

Scanned with OKEN Scanner

5. Add parameters to allow filtering (e.g., date range, temple name).

6. Generate and preview the report to verify the data accuracy and presentation.

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

9/10/25

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.

VELTECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF SCIENCE AND TECHNOLOGY

DATABASE MANAGEMENT SYSTEMS

Course Code: 10211CS207

Task 12 (Micro Project)

Use Case 3 – GIFT COUPON APPLICATION

Implemented by: CHANDRAJIT KAMALAKKANNAN (VTU28548)

Team Members:

Name	VTU Number
CHANDRAJIT KAMALAKKANNAN	VTU28548
SUNKESWARA SUSHANTH	VTU30148
SHAIK GOUSE MOHIDDIN	VTU29776
DODLA SANTHOSH	VTU30639
MANNURU MEDHINI	VTU30464
KALLURI PEDDA BABU	VTU29154
VADAMADHURA DILEEP KUMAR	VTU275557
YARASANI TARUN KUMAR REDDY	VTU30548
KOMMINA SRUTHI	VTU30547
GOSU BHARGAV RAM	VTU30538
SHAIK SHAHIL ROHAN	VTU27624
THONDAPU SAI JASWANTH	VTU28996
DASARI KOTISURYA	VTU28657

Scanned with OKEN Scanner

Link to MyASUS for easy file transfer

Physical webcam privacy shield

Smooth mega-sized touchpad

RAVURI NAGA DURGA PRASAD	VTU27974
DADIPINENI BHARGAVI	VTU29377
PANAGANTI SAI SRINATH	VTU27736
CHAMARTHI TEJESWAR RAJU	VTU28933
NAGANABOYINA MOHAN VENKAT	VTU30539
KANUPARTHI POOJITH REDDY	VTU29344
KONDAPATURI MADHU SUMANTH	VTU27547
KUCHANA SRAVIKA	VTU29764
VEPURALA VIPIN RAJ	VTU29362
GANAPATHIRAJU JAGANNADHA VARMA	VTU27789
KODURU KESAVA	VTU28867
GANDLURU VENKATA ARAVIND	VTU29367

Scanned with OKEN Scanner

Abstract

The Gift Coupon Application is a transactional system that enables users to manage, redeem, and purchase gift coupons securely. The application integrates offers, user accounts, and payment handling in a single relational database system. To ensure strong consistency, the system employs a relational database (MySQL/PostgreSQL) that maintains ACID properties while scaling horizontally. This ensures transactional integrity during concurrent coupon redemptions and payments. The system is normalized to Third Normal Form (3NF) to reduce redundancy and maintain data accuracy.

Aim

To develop a **Gift Coupon Application** that handles offers, coupon generation, and payment transactions efficiently using a **relational database system** that maintains **ACID properties** and supports **horizontal scalability**.

Introduction

Gift coupons and discount offers have become integral to modern e-commerce platforms.
Businesses use these systems to attract customers and retain loyalty.
However, handling large-scale transactional data (coupon creation, redemption, and payments) requires a database capable of strong consistency and atomic operations.
This project implements a Gift Coupon Management System using a **relational database** for:

- Reliable storage of transactional data
- Fast lookups of offers and coupons
- Maintaining normalization and consistency

The relational database ensures that all coupon redemption and payment operations are secure, atomic, and deadlock-free under normal conditions.

System Requirements

Hardware Requirements

Component	Specification
Processor	Intel i3 or higher

Scanned with OKEN Scanner

Component	Specification
RAM	4 GB or more
Storage	Minimum 500 MB
Display	1024×768 resolution

Software Requirements

Component	Specification
Operating System	Windows / Linux
Frontend	HTML, CSS, JavaScript
Backend	Python / PHP / Java
Database	MySQL / PostgreSQL
Tools	VS Code / MySQL Workbench

Existing System

- The existing coupon systems store data in **flat files or non-relational databases**.
- These systems lack **strong ACID guarantees**.
- They often face **inconsistency issues** when multiple users redeem the same coupon simultaneously.
- Manual validation of coupon status is required.

Disadvantages

- No guarantee of data integrity
- Possible duplicate coupon redemption
- No proper transaction rollback in case of failures

Proposed System

The proposed system uses a **Relational Database Management System (RDBMS)** such as **MySQL** for:

- Maintaining **ACID transactions** during coupon creation, redemption, and payment.
- Implementing **3NF normalization** to ensure data integrity.
- Enabling **horizontal scalability** through database clustering.
- Preventing deadlocks with proper transaction management and indexing.

Advantages

- Ensures data accuracy and atomicity.

Scanned with OKEN Scanner

- Prevents multiple redemptions of the same coupon.
- Provides real-time offer tracking.
- Secure and scalable for enterprise use.

ER Diagram

Entities:

- **User** (User_ID, Name, Email, Password)
- **Coupon** (Coupon_ID, Code, Discount, Expiry_Date, Status)
- **Offer** (Offer_ID, Description, Valid_From, Valid_To)
- **Payment** (Payment_ID, User_ID, Amount, Date, Status)
- **CouponUsage** (Usage_ID, User_ID, Coupon_ID, Payment_ID, Usage_Date)

Relationships:

- A *User* can redeem multiple *Coupons*.
- A *Coupon* may belong to one *Offer*.
- A *Payment* is associated with one *User* and one *CouponUsage*.

(Visual ER diagram description — can be drawn in draw.io or MySQL Workbench)

Use Case Diagram

Actors:

- User
- Admin
- Payment Gateway

Use Cases:

- Register/Login
- View Offers
- Generate Coupon
- Redeem Coupon
- Make Payment
- Manage Offers (Admin)

(Draw the use case diagram showing relationships between actors and use cases — Admin manages offers, User redeems coupons, Payment Gateway handles payments.)

Scanned with OKEN Scanner

Implementation

The system implementation consists of:

1. **Frontend:** Web interface for users to view and redeem coupons.
2. **Backend:** REST API or server-side logic for handling transactions.
3. **Database:** MySQL for ACID-compliant transaction processing.

Database Tables (Normalized up to 3NF):

Users Table

```
CREATE TABLE Users (
  User_ID INT PRIMARY KEY AUTO_INCREMENT,
  Name VARCHAR(100),
  Email VARCHAR(100) UNIQUE,
  Password VARCHAR(100)
);
```

Coupons Table

```
CREATE TABLE Coupons (
  Coupon_ID INT PRIMARY KEY AUTO_INCREMENT,
  Code VARCHAR(50) UNIQUE,
  Discount DECIMAL(5,2),
  Expiry_Date DATE,
  Status VARCHAR(20)
);
```

Offers Table

```
CREATE TABLE Offers (
  Offer_ID INT PRIMARY KEY AUTO_INCREMENT,
  Description VARCHAR(255),
  Valid_From DATE,
  Valid_To DATE
);
```

Payments Table

```
CREATE TABLE Payments (
  Payment_ID INT PRIMARY KEY AUTO_INCREMENT,
  User_ID INT,
  Amount DECIMAL(10,2),
  Date DATETIME,
  Status VARCHAR(20),
  FOREIGN KEY (User_ID) REFERENCES Users(User_ID)
);
```

Scanned with OKEN Scanner

CouponUsage Table

```
CREATE TABLE CouponUsage (
  Usage_ID INT PRIMARY KEY AUTO_INCREMENT,
  User_ID INT,
  Coupon_ID INT,
  Payment_ID INT,
  Usage_Date DATETIME,
  FOREIGN KEY (User_ID) REFERENCES Users(User_ID),
  FOREIGN KEY (Coupon_ID) REFERENCES Coupons(Coupon_ID),
  FOREIGN KEY (Payment_ID) REFERENCES Payments(Payment_ID)
);
```

Code (Sample Implementation in Python + MySQL)

```
import mysql.connector
from datetime import datetime

# Database connection
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    database="gift_coupon_db"
)
cursor = conn.cursor()

# Redeem coupon function
def redeem_coupon(user_id, coupon_code, amount):
    try:
        conn.start_transaction()

        # Check coupon validity
        cursor.execute("SELECT Coupon_ID, Discount, Status FROM Coupons WHERE
Code=%s", (coupon_code,))
        result = cursor.fetchone()
        if not result or result[2] != 'ACTIVE':
            print("Invalid or expired coupon!")
            return

        coupon_id, discount, status = result
        final_amount = amount - (amount * discount / 100)

        # Insert payment record
        cursor.execute("INSERT INTO Payments (User_ID, Amount, Date, Status)
VALUES (%s, %s, %s, %s)",
                       (user_id, final_amount, datetime.now(), 'SUCCESS'))
        payment_id = cursor.lastrowid

        # Record coupon usage
        cursor.execute("INSERT INTO CouponUsage (User_ID, Coupon_ID,
Payment_ID, Usage_Date) VALUES (%s, %s, %s, %s)",
                       (user_id, coupon_id, payment_id, datetime.now()))
```

Scanned with OKEN Scanner

```
        # Update coupon status
        cursor.execute("UPDATE Coupons SET Status='USED' WHERE Coupon_ID=%s",
(coupon_id,))

        conn.commit()
        print("Coupon redeemed successfully!")

    except Exception as e:
        conn.rollback()
        print("Transaction failed:", e)

# Example usage
redeem_coupon(1, "OFFER2025", 500)
```

Conclusion

The Gift Coupon Application effectively demonstrates how **relational databases** ensure **transaction integrity** and **data consistency** for coupon and payment management systems. By following **3NF normalization** and using **ACID-compliant transactions**, the system prevents duplicate coupon redemption and payment mismatches.
The project successfully shows that **horizontal scalability with clustering** and **relational normalization** can coexist, making it suitable for large-scale enterprise deployments.

16/10/25

EXP NO	02
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	18
DATE	

Scanned with OKEN Scanner