Vel Tech

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: CHANDRAJEER KAMALAKKANNAN

VTU.NO: 28548

REG.NO: Q4UECL0009

BRANCH: CSE (AIML)

YEAR/SEM: 2nd year, III sem

SLOT: S10 L13

Vel Tech
Rangarajan Dr. Sagunthala
R&D Institute of Science and Technology

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

ACADEMIC YEAR 2025-26 (SUMMER SEMESTER)

BONAFIDE CERTIFICATE

NAME : CHANDRAJIT K
BRANCH : CSE (AIML)
VTU NO. : 28548
REG.NO. : 24UECL0009
YEAR/SEM : 2nd year III SEM
SLOT NO. : S10 L13

Certified that this is a bonafide record of work done by above student in the **"10211CA207-DATABASE MANAGEMENT SYSTEMS LABORATORY"** during the year 2025-2026 (Summer Semester).

R.T. 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**

Name- Chandrajit.k

INDEX

Date	Title	Page No.	Marks	Faculty Signature
24/07/25	Conceptual Design after Formal Technical Review	2-8	18	24/7
31/07/25	Generating design of other traditional database models	10-26	18	31/7
07/08/25	Developing queries with DML Single-row functions and operations	28-36	18	7/8
14/08/25	Developing queries with DML Multi-row functions and operations	38-46	18	14/8
21/08/25	Writing Join Queries, equivalent, and/or recursive queries	48-54	18	21/8
28/08/25	Writing PL/SQL using Procedures, Function	56-62	20	28/8
04/09/25	Writing PL/SQL using Loops	64-70	20	4/9
11/09/25	Normalizing databases using functional dependencies up to BCNF	72-78	18	11/9
18/09/25	Backing up and recovery in databases	80-86	18	18/9
25/09/25	CRUD operations in Document databases	88-90	19	25/9
09/10/25	CRUD operations in Graph databases	92-94	19	9/10
16/10/25	Micro Project: Grift coupon application completed	96-104	18	16/10

Total Marks: 222/240

Signature of Faculty

04/07/25 Task - 1 Conceptual Design after full text review

A Temple ticket online booking management system

A temple ticket online booking management system devoles to book tickets for temple visits, special darshan . poojas and other event online, reducing physical queues and improving crowd management. These system often include textures like date and time slot selection, payment processing and the generation of tickets or passes.

Entity:

The real world concept that can be distinctly identified . ex - included students, courses or products.

Entity set:

A collection of entities of the same type for instance all students in a university world from an entity test.

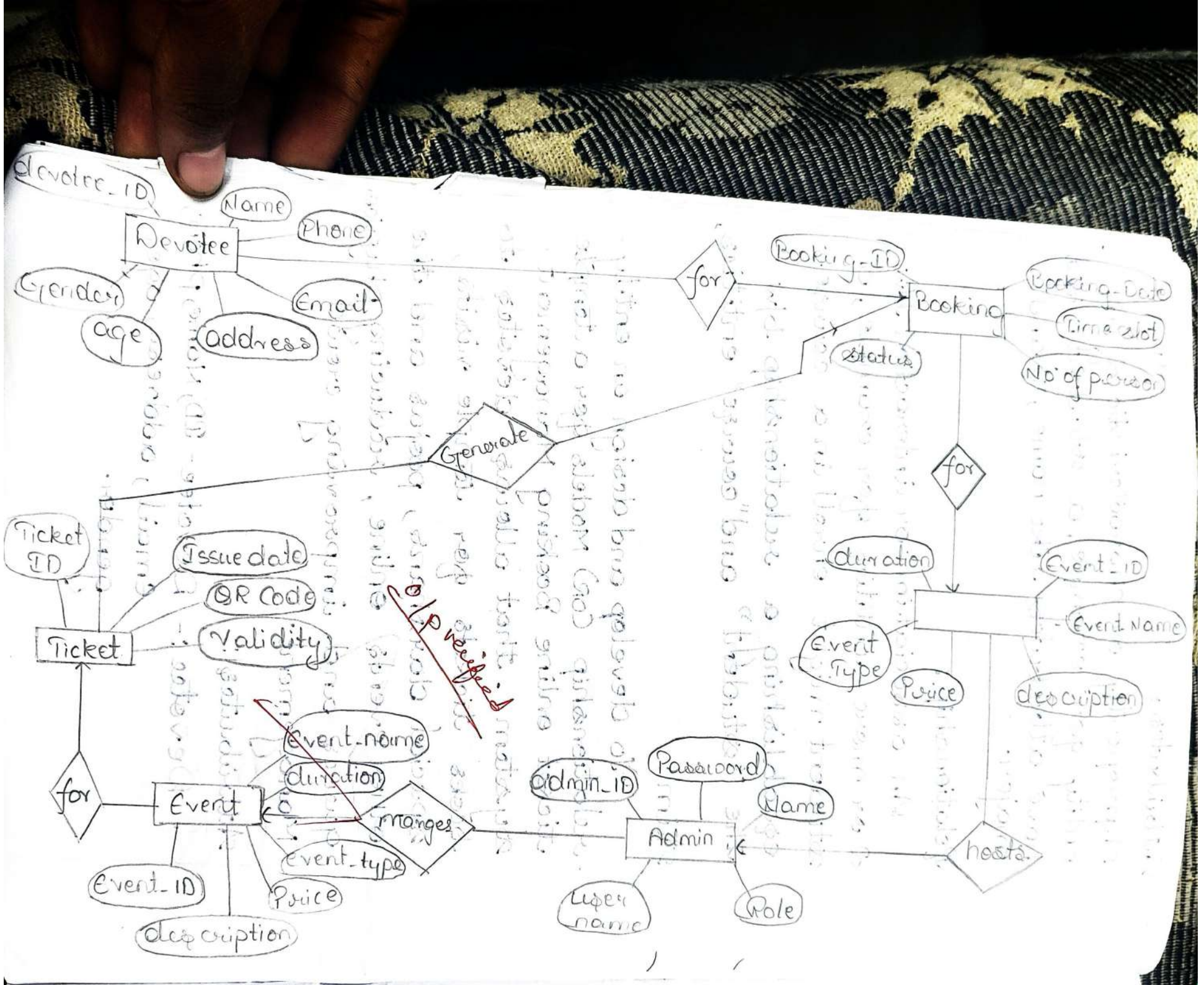
Attributes:
A property or characteristic of an entity for example, a student might have attributes like name, ID and major.

Relationship:
An association or interaction b/w 2 or more entities for example, a student might enroll in a course. establishing a relationship b/w the "student" and "course" entities.

Aim:
To develop and design an entity-relationship (ER) Models for a temple ticket online Booking Management system that allows devotes to books tickets for temple visits, special darshans, poojas and the other events online, reducing physical queues and improving crowd Management.

Attributes:

- Devotee:- Devotee-ID, Name, phone, Email, address, age, gender.


devotee_ID
Name
Phone
Devotee
Gender
age
Address
Email
for
Booking_ID
Booking
Booking_Date
Time slot
No of person
Status
Generate
for
Ticket ID
Issue date
QR Code
Validity
Ticket
duration
Event_ID
Event Name
description
Event Type
Price
for
Event
Event_name
duration
manages
Event_type
Price
description
Event_ID
admin_ID
Password
Name
Admin
User name
Role
hosts

- Booking :- Booking-ID, Date, Time-slot, No. of person, status.
- Event: Event_ID. Event_Name, Description price, Event_type, duration.
- Temple: Temple_ID, Temple-name, Location, opening_hours.
- payment: payment_ID, amount, Method, status, Date,
- Ticket :- Ticket_ID, issue-Date, QR-code, validity
- admin: Admin_ID, Name, username, password, Role.

Types of attributes:

- Simple: Name, price
- composite: address (street, city, stat
- 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-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)	3
RECORD (5)	5
TOTAL (20)	5+5+5+3=18
SIGN WITH DATE	

29/7/25

Result:

The ER diagram for the temple ticket Booking system was successfully designed, showing all entities, attributes, and relationship with correct cordinalities for database implementations.

31/7/25

Task - 2 Generating design of other traditional database model

Creating Hierarchical / Network model of the database by enhancing the sound abstract data by performing following tasks using form of inheritance.

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

2.b check is-a hierarchy/has-a-hierarchy and performs generaliza-tion and/or specializtion relationsh-ip.

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

2.d Rename the relations.

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

Aim:

Creating Hierarchical / Network model of the database by enhancing the found abstract data by performing following tasks using forms of inheritance.

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

- Relationship1: Temple conducts pooja (one to many)
- specificity - A devotee can book many tickets, but each ticket booking belongs to only one devotee
- Surplus Relation: No surplus relation is needed for this relationship since it is already many-to-one.

Relationship2: Ticket belongs to pooja (many to one)

- specificity: one temple can conduct many poojas, but each pooja ticket booking belongs to only one devotee.

Surplus Relation: No surplus relation needed, one to many.

Relationship3: Customer book ticket (many-to-many)

specificity: A customer can book many tickets, and a Ticket can be booked by many customers (for group booking)

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

Relationship4: 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 relationship is sufficient

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

Generalization Example:

- Entities: Customer, Admin
- Common attributes: User_ID, First_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_services)

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 with valid range	CHECK (seat_number Between 1 AND 500)

Optional

2.d Rename the Relations (tables)

Example renaming columns for clarity:

```
ALTER TABLE customer RENAME COLUMN
  contact_no TO phone_no;

ALTER TABLE Ticket RENAME COLUMN price to
  p ticket_price;

ALTER TABLE Booking RENAME COLUMN
  book_date TO booking_date;
```

2.e Perform SQL Relations using DDL and DCL commands:

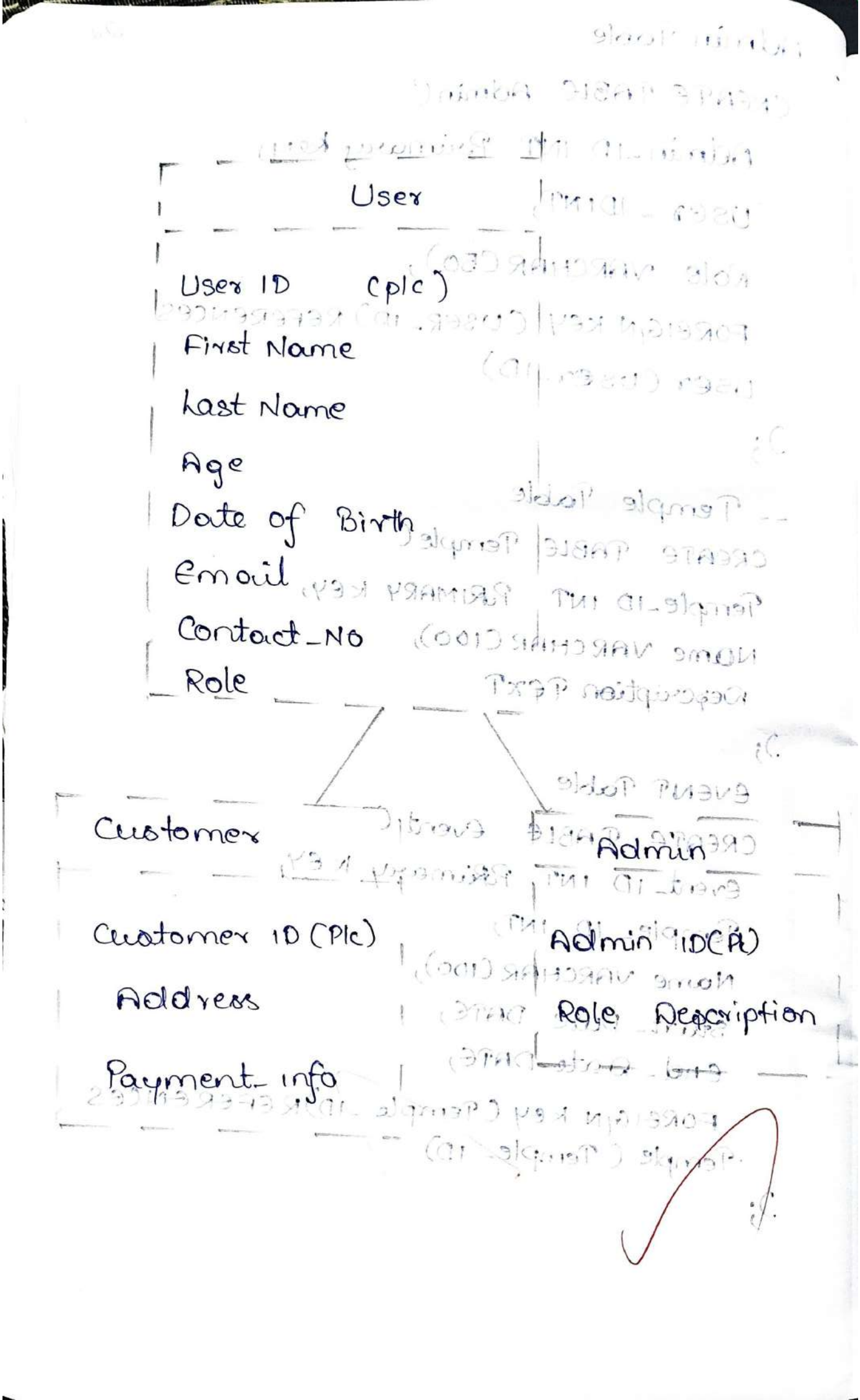
DDL (Data Definition Language):

```
-- Create User Table (Generalization)
CREATE TABLE USER(
User_ID INT Primary Key,
first_Name VarcHAR(50),
LAST_Name VARCHAR(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_IDINT,
AGe INT,
Address VARCHAR(255),
Payment_info VARCHAR(255),
FOREIGN KEY (USER_ID) REFERENCES
User (USER_ID)
);
```

```
-- Admin Table
CREATE TABLE Admin(
  Admin_ID INT Primary key,
  USER_IDINT,
  Role VARCHAR(50),
  FOREIGN KEY (USER_ID) REFERENCES
  USER (USER_ID)
);

-- Temple Table
 CREATE TABLE Temple(
 Temple_ID INT PRIMARY KEY,
 NAme 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)
);
```

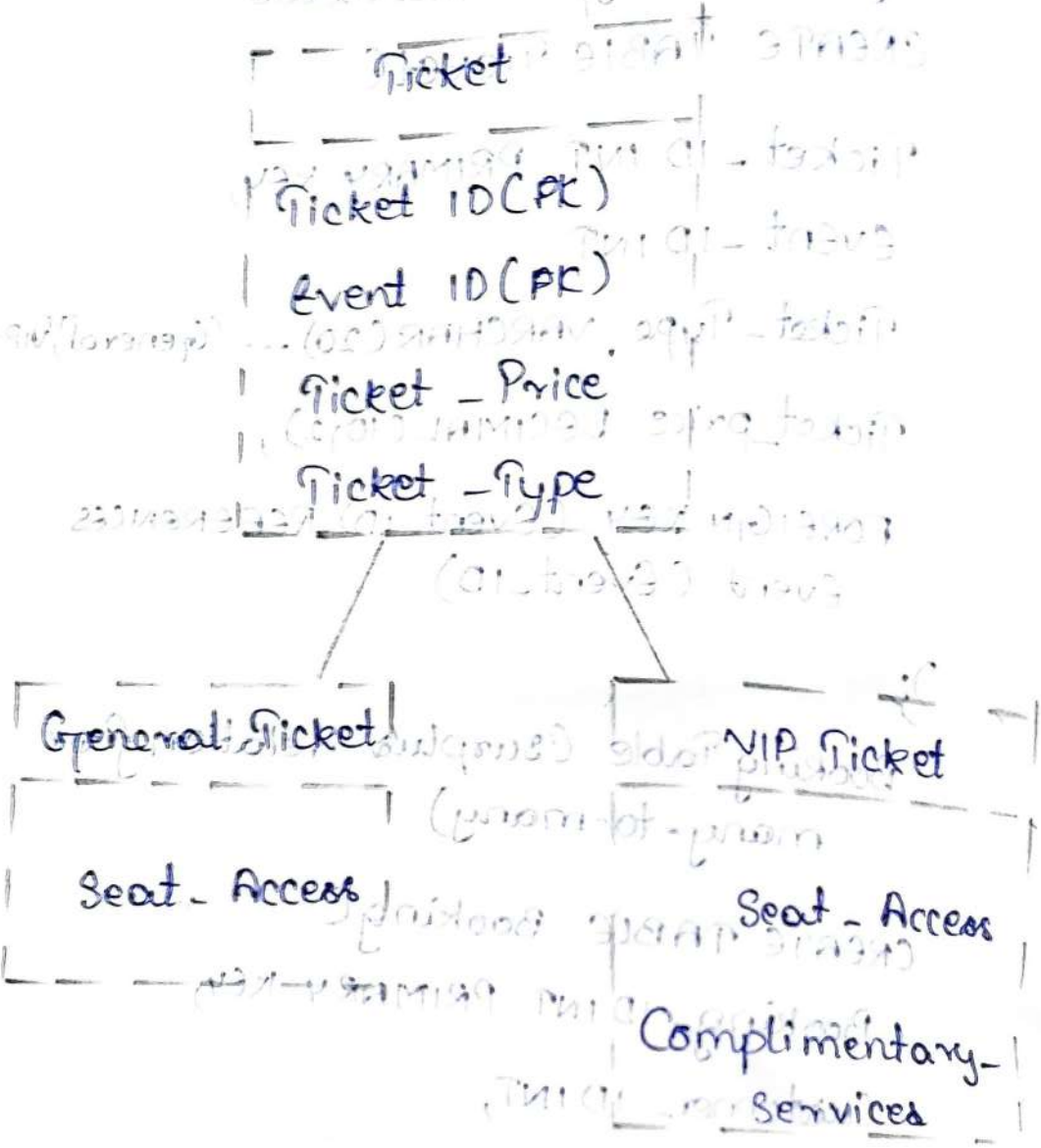

User
User ID (plc)
First Name
Last Name
Age
Date of Birth
Email
Contact_No
Role
Customer
Customer ID (Plc)
Address
Payment_info
Admin
Admin ID(PK)
Role Description

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),
    FOREIGN KEY (Event_ID) REFERENCES
      Event (Event_ID)
);
```

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

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


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

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

Result

Thus the hierarchial model and network model for online Ticket Booking Managment System has been successfully created.

7/8/25

Task-3

Using clauses, operators and function in queries.

Aim: To perform query processing on a Temple tickets online Booking management system for different retrieval results of queries using DML, DRL operations with aggregate functions, data function, string function, Bet clauses and operators.

Temple

Temple ID	Name	location	Contact no°
TID 01	Tirumalar venkateshwar	Tirupati	9014276954
TID02	Meenakshi amman	Madurai	9798 922278
TID03	Baashi Vish-woarth	Varanasi	7013982716
TID04	Jaganath Temple	Puri	9701224404
TID05	Golden Temple	Amritsar	9908616499

Visitor

Visitor ID	FName	LName	age	Email	Contact
V001	Anil	Kumar	30	anil@gm -ail.com	9014276594
V002	Akash	Reddy	25	aku@gm -ail.com	970122 4404
V003	Priya	Sharma	28	Riyar@gm -ail.com	990829 0010
V004	Aarav	Patel	35	Aarav@gm -ail.com	7013987 216
V005	Sneha	Rao	22	Sneharr@gm -ail.com	798922 -2725

Booking

Booking ID	TempleID	visitorID	Booking date	Ticket type	amount
B001	TID01	V001	2024-6-15	VIP	500
B002	TID02	V002	2024-06-16	Gen -eral	100
B003	TID01	V003	2024-06-17	VIP	700
B004	TID03	V004	2024-06-18	Gene VIPral	100
B005	TID01	V005	2024-06-19	General	200

O/P verified

Priest

Priest ID	FName	LName	Age	Email	Contact-no
P001	Ramesh	Iyer	50	ramesh@gmail.com	90 14276 -954
P002	Suresh	Sharma	45	suresh@gmail.com	97012294 04
P003	Manish	Das	40	manish@gmail.com	7959277 285

Retrieve details of visitors whole first name starts within

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

Result:

VisitorID	FName	LName	Age	Contact_No
V002	Akash	Reddy	25	9701224404
V004	Aarav	Patel	35	7013982716

3. Add a Column for special - seva in Booking Table

```
ALTER TABLE Booking ADD special SEVA
VARCHAR(50);
```

Result;

Table altered successfully

4. count the number of VIP ticket booking.

```
SELECT COUNT(*)
FROM BOOKING
WHERE Ticket-type = 'VIP';
```

Result

Count(*)

5. Display temple detail for Temple IDS 'TIDO1', 'TIDO3', and 'TIDO4';

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

Result:-

Temple ID	Name	location	Contact_NO
TIDO1	Tirumala	Tirupati	9701224404
TIDO2	Kashi	Varanasi	7013952716
TIDO3	Goldem 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	FName	LName
V005	Sneha	Rao

7. Find the priest ID of priest who have not been assigned any temple.

```
SELECT priest ID
FROM priest
WHERE Temple ID is NULL;
```

Result;

Priest ID

CNO Results : All priests are assigned.

VEL TECH - CSE

EX NO	3
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	2
RECORD (5)	+5
TOTAL (20)	12+5=17
SIGN WITH DATE	

Results:- Thus, query processing for temple ticket online Booking Management system using clauses. Operators, and function was successfully performed.

14/8/25 Task-4:

Using functions in queries and writing Sub queries :-

Aim : To perform advanced query processing and test its heuristics by designing optimal correlated and Nested subqueries such as finding summary statics in temple ticket online Booking Managment system.

4.1 To retrive 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;
```

Output :-

Temple ID	TempleName	location	Total Bookings
TID01	Tirumala	Tirupati	3
TID02	Meenakshi Amman	madurai	1
TID03	Kashi	Varanasi	1
TID04	Jagannath temple	Puri	0
TID05	Golden Temple	Amritsar	0

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.Tempeled = b.TempleID
WHERE b.Ticket_Type = 'special'
Group By t.Name;
```

Output

Temple Name.

Tirumala Venkateshwara

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

```
SELECT *
FROM Temple
WHERE TempleID DNC
  SELECT TempleID
  FROM Booking
WHERE Ticket_Type = (VIP)
);
```

Output

Temple ID	Name	location	Contact_No
TID01	Trimula	Tirupati	9701224404
TID03	Kashi	Varanasi	7013982716

4:4 To retrieve visitors and booking details of visitors who are above 25 years old.

```
SELET V.visitor ID,
      V.FName As visitor Name,
      V.age
      V.BookingID
      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 ID	visitorName	Age	Booking ID	Amount
V001	Anil	30	B001	500
V003	Priya	28	B003	100
V004	Aarav	35	B004	700

4.5 To retrive the details of temples with no bookings.

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

Output

Temple ID	Name	Location	Contact_No
TID04	Jagannath Temple	Puri	9701224404
TID05	GoldenTemple	Amritsar	9014276954

4.6 To retrieve the templeid, Templename, and visitor name for a particular visitor id given

```
SELECT t.Temple ID,
       t.Name AS Temple Name,
       v.FName AS visitor Name
FROM Temple t.
  Join Booking t
  ON t. Temple ID = b. Temple ID
```

```
Joint visitor V
  ON b. visitorID = V.visitorID
  WHERE V.visitorID = ('V005');
```

Output :-

Temple ID	Temple Name	visitor Name
TID01	Tirumala	Sonika

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

14/8

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

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

BookingID	Devotee	PoojaName	BooDate	Slottime	Status
B101	Rajesh	Superbhata seva	22-Jun-23	06:00AM	Confirmed
B102	Meena	Archana	22-Jun-2023	09:00AM	Confirmed
B103	Arjun	Abhishekam	23-Jun-2023	07:30AM	Cancelled
B104	Kavitha	special Darshan	23-Jun-2023	08:30AM	Confirmed
B105	Suresh	Abhishekam	24-Jun-2023	06:30AM	Confirmed

21/8/25 Task 5

Writing Join Queries, equivalent, and /or recursive queries:

Aim: To perform advanced query processing and tests its heuristic using Join Queries, equivalent queries and recursive queries for Temple Ticket online Booking Management system, which manages devotees, priests, poojas tickets and booking details.

5.1 To retrieve all temples and their poojas.

```
SELECT t.Name AS Temple, p.poojaName,
p.price
FROM Temple t
JOIN Pooja p ON t.TempleID = p.TempleID;
```

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

```
SELECT b.BookingID, d.FName AS Devotee,
  p.PoojaName, b.BookingDate, b.slotTime,
  b.Status
FROM Booking b
JOIN Devotee d ON b.DevoteeID = d.DevoteeID
JOIN Pooja p ON b.poojaID = p.PoojaID
```

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

DevoteeID	FName	MobileNO
D103	Arjun	9876543210
D105	Suresh	8765432190

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

5.3 Count the number of bookings made by each devotee

```
SELECT d.FName AS Devotee, COUNT(b.BookingID)
    AS TotalBookings
FROM Devotee d
LEFT JOIN Booking b ON d.devoteeID = b.Devot-
    --eeID
GROUP By d.FName;
```

5.4 To find all devotees who booked 'Abhishekam'

```
SELECT d.DevoteeID, d.FName, d.MobileNO
FROM Devotee d
JOIN Pooja p ON b.poojaID = p.PoojaID
WHERE p.poojaName = 'Abhishekam';
```

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

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

Temple	Cancelled Bookings
Kanchi Temple	1

TempleID	Name	location	Head Priest
T001	Tirupati Temple	Tirupati	Ramanujam
T002	Kanchi Temple	Kanchipuram	Narayan Das
T003	Madurai Temple	Madurai	Subramani-an

FName	Age	BookingID	BooDate	Slot time	Stati
Suresh	55	B105	24-Jun-2023	06:30 AM	Conf-irme-d

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

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

5.7 To retrieve temple details where booking were succesful

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

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

```
SELECT d.FName, d.Age, b.BookingID, b.BookingDate,
    b.SlotTime, b.Status
FROM Devotee d
JOIN Booking b ON d.DevoteeID = b.DevoteeID
WHERE d.Age > 50;
```

Temple ID	Name	location	Head Priest
T004	Rameshwa-ram Temple	Ramesh-wa-ram	Vishnu Devan

Temple	Pooja Name	Devotee	Booking Date	Slot Time
Tirupati Temple	Archana	Meena	22-Jun-2023	09:00 AM

0/P

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

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

5:10 To retrieve temple, pooja and devotee details for given Booking ID

```
SELECT t.Name AS Temple, p.PoojaName, d.FName,
       AS Devotee, b.BookingDate, b.SlotTime
FROM Booking b
JOIN Temple t ON b.TempleID = t.TempleID
JOIN Pooja p ON . b. PoojaID = p.poojaID
JOIN Devotee d ON b.DevoteeID = d.DevoteeID
WHERE b.BookingID = 8602;
```

VEL TECH - CSE

EX NO:02	5
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	13+5=18
SIGN WITH DATE	

all queries

Result:

Thus, the Temple Ticket online Booking Management System queries using Join queries, equivalent queries, and recursive queries were executed successfully to manage temples, devotees bookings.

28/8/25 Task 6

Programming using PL/SQL

Procedure, functions and loops on Number theory and business scenarios like.

Aim: To develop a PL/SQL program simulating an online booking system for temple tickets where devotees can book multiple tickets, total cost is calculated with discounts applied, and multiple bookings are processed using procedures, functions and loops.

Algorithm:

1. start
2. Define ticket price (e.g Rs 50)
3. For each booking request (devotee), input the number of tickets.
4. Calculate total price = ticket price x number of tickets.
5. Apply discount:
   - 10 or more tickets → 10% discount
   - 5 to 9 tickets → 5% discount
   - Below 5 tickets → no discount
6. Calculate final amount after discount.

7. Store or display booking details
8. Use loop to handle multiple booking requests
9. End

Source Code:

```
-- Function to calculate discounted amount based on tickets

  CREATE OR REPLACE FUNCTIONS
calculate_final_amount (tickets IN NUMBER)
 RETURN NUMBER IS
      ticket_price NUMBER := 50;
      total_price NUMBER;
      discount_rate Number; = 0;
      final_price NUMBER;
BEGIN
     total_price := ticket_price
* tickets;
    IF tickets >= 10 THEN
        discount_rate := 0.10; --10%.disco-
                                  -unt
    ELSEIF tickets >=5 THEN
        discounts_rate := 0.05; -- 5%.
                                  discount
    ENDIF;

    final_price := total_price - (total_price
                  * discount_rate);
    RETURN final_price;
    END;
    /
```

-- Procedure to book tickets and print booking details

```
CREATE OR REPLACE PROCEDURE
book_temple_ticket (devotee_id IN
NUMBER, tickets IN NUMBER) IS
    amount NUMBER;
BEGIN
    amount :=
calculate_final_amount(tickets);
    DBMS_OUTPUT.PUT_LINE('Devotee ID:
    ' || devotee_id);
    DBMS_OUTPUT.PUT_LINE('Tickets
Booked: ' || tickets);
    DBMS_OUTPUT.PUT_LINE('AMOUNT to
Pay (after discount): Rs. ' || amount);
END;
/
-- Anonymous block to perform multiple
bookings (simulate online requests)
DECLARE
    TYPE BookingList IS TABLE OF NUMBER
INDEX BY PLS_INTEGER;
    devotees BookingList;
    tickets_list BookingList;
BEGIN
-- SAMPLE data: devotees and tickets
    booked
    devotees(1) := 101;
    tickets_list(1) := 3;
```

Output

Devotee ID: 101

Tickets Booked: 3

Amount to Pay (after discount): Rs 150

Devotee ID: 102

Ticket Booked: 6

Amount to Pay (after discount): Rs 285

Devotee ID: 108

Tickets Booked: 12

Amount to Pay (after discount): Rs 540

```
devotees(2) := 102;
tickets_List(2) := 6;
devotees(3) := 103;
ticket_List(3) := 12;

For i In 1.. devotees.COUNT LOOP
book_temple_ticket(devotees(i),
tickets_List(i));
        DBMS_OUTPUT.PUT_LINE('----------
---------');
    END LOOP;
END;
/
```

VEL TECH - CSE	
EX NO	6
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	+5
TOTAL (20)	1×+5=20
SIGN WITH DATE	G.M.Thiru 04.09.24

Result: Thus, the Temple Ticket online Booking Managment system using PL/SQL Procedure, functions and loops on Number theory and buisness scenarios is successfully completed.

4/9/25 Task 7

Triggers, Views and Exception

Aim:

To conduct events, views, and exception handling on CRUD operations for temple Ticket Online Booking Managment System.

a) Trigger

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

Sql

```
CREATE OR REPLACE TRIGGER
trg_insert_ticket
AFTER INSERT ON Booking
FOR EACH ~~NOW~~ ROW
BEGIN
    INSERT INTO TICKET (TICKETID,
BookingID, DevoteeNew, Age, Gender,
QR code, Status)
 VALUES (
    seq_ticket.NEXTVAL,
    :NEW.BookingID,
    'Unknown',
    NULL,
```

```
    NULL,
    SYS_GUID(),
    'ACTIVE'
  );
END;
/
```

b) View

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

Sql

```
CREATE OR REPLACE VIEW
AS
SELECT
    b.BookingID,
    b.BookingRef,
    u.Name AS Devotee,
    t.Name AS Temple
    dt.Name As DarshanType,
    s.StartTs As SlotStart,
    s.EndTs AS SlotEnd,
    b.Qty,
    b.Amount,
    b.Status
FROM Booking b
JOIN Users u ON b.UserID = u.UserID
JOIN Temple t ON b.TempleID = t.TempleID
JOIN Slot s ON b.SlotID = s.slotID
```

```
JOIN DarshanType dt ON s.DarshanType
ID= dt.DarshanTypeID;

SELECT * FROM BookingDetailsView;
```

c) Non-Recursive PL/SQL Procedure

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

SQL

```
CREATE OR REPLACE PROCEDURE
Get Even BookingIDs For temple(
  in_temple_id NUMBER
  out_even_booking_ids OUT
SYS.ODCINUMBERLIST
)AS
BEGIN
    out_even_booking_ids := SYS.ODCINUM
                            BER LIST();

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

```
DECLARE
   temple_id NUMBER := 101;
   even_booking_ids SYS.ODCINUMBERLIST;
BEGIN
   GetEvenBookingIDsForTemple(temple_id,
even_booking_ids);
   FOR i IN 1..even_booking_ids.COUNT LOOP
      DBMS_OUTPUT.      ('Even
BookingID: ' || even_booking_ids(i));
temple_id NUMBER := 101;
   even_booking_ids SYS.ODCINUMBERLIST;
BEGIN
   GetEvenBookingIDsForTemple(temple_
id, even_booking_ids);
   FOR i IN 1... even_booking_ids.COUNT LOOP
      DBMS_OUTPUT.PUT_LINE('Even
BookingID: ' || even_booking_ids(i));
   END LOOP;
END;
/
```

VEL TECH - CSE	
EX NO	7
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	15+5=20
SIGN WITH DATE	T.M.Thejr 04.09.24

Result;

Thus, the Triggers, views and Exceptions for the temple Ticket Online Booking Management System were successfully implemented and verified.

11/9/25

Task 8

CRUD operations in Document database

Aim:

To design and interact with a MongoDB document database using Mongoose (a MongoDB ODM) via Node.js and NPM, and to perform basic CRUD operations:

Algorithm

1. Install dependencies: Initialize a Node.js project and install mongoose.
2. Connect to MongoDB: Use Mongoose to connect to a local or cloud Mongo DB database.
3. Define a schema: Create a Mongoose schema, for the documents (collection structure).
4. Create a Model: Based on the schema, define a model to interact with the collection.
5. Perform CRUD operations:
   - Create / Insert: add new documents
   - Read / Find: Query documents based on filters
   - Update: Modify existing documents
   - Delete / Remove: Remove documents from the collection.

1. Initialize Project and Install Mongoose

js

```
const mongoose = require('mongoose');
// step 1: Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/
                  myDB', {
    useNewUrlParser: true,
    useUnifiedTopology: true
}).then(() => {
    console.log("Connected to MongoDB");
}).catch(error("MongoDB connection error",
                  err));
});
// step 2: Define a Schema
    const ticketUserSchema = new mongoose.Schema({
        name: String,
VisitDate: String,
ticketType: Number
});
// step 3: Create a model
  const User = mongoose.model('User', userSchema);
// CREATE: Insert a new user
  const createTicketUser = async () => {
    const user = new User({
        name: "Alice",
visitDate: "alice@example.com",
ticketType: "VIP"
});
```

```
// Read: Get all users
const getUsers = async () => {
    const users = await User.
await ticket.save();
console.log("Ticket Created:", ticket);
};

// Read
const getTickets = async () => {
    const tickets = await Ticket.find();
    console.log("All Tickets:", tickets);
};

// UPDATE
const updateTicket = async (id) => {
    const update = await Ticket.findById
    AndUpdate(id, { ticketType: "General" },
    { new: true });
    console.log("Update ticket:", updated);
};

// DELETE
const deleteTicket = async (id) => {
    await Ticket.findById AndDelete(id);
    console.log(`Ticket Deleted:`, id);
};
```

How to Run

1. start mongo DB (local)
2. In terminal.

```
node app.js
```

Output

Ticket Created: {
_id : "65012 3abc..."),
name : "Sita Devi"),
visitDate : "2025-09-15"),
ticketType : "VIP"),
__v : 0
}

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

4/9/25

Result:

Thus the program to design & interact with MongoDB document database using Mangoose (a MongoDB ODM) via Node.js and NPM, and to perform basic CRUD operations.

18/9/25

Task 9

CRUD operations in Graph database

Aim: To design a graph database model for temple ticket booking using Neo4j and GraphQL and perform CRUD operations such as creating, inserting, querying, finding, and deleting graph data.

Algorithm:

1. Define graph entities (nodes) - Temple, User, Ticket.
2. Define relationships - User Booked_For Ticket, Ticket FOR_TEMPLE Temple.
3. Design GraphQL schema types for these entities with relationship.
4. Use Cypher queries for CRUD operations:
   - Create nodes and relationships.
   - Insert data into nodes and Link with relationships.
   - Query graph data by matching nodes and relationships.
   - Find specific nodes/relationships with filters.
   - Delete nodes and detach relation-ship.

5. Expose GraphQL API using Neo4j GraphQL integration for client-side CRUD operation interaction.

Source Code

```
type Temple {
    templeID: ID!
    name: String!
    location: String!
    tickets: [Ticket!]! @relationship
        (type: "FOR_TEMPLE";
        direction: IN)}

type User {
  userID: ID!
  name: String!
  email: String!
  tickets: [Ticket!]! @relationship(type:
        "BOOKED_FOR", direction: OUT)}

type Ticket {
  ticketID: ID!
  date: String!
  seatNumber: String!
  Status: String!
  user: User! @relationship (type: "Booked
        FOR", direction: IN)
  temple: Temple! @relationship (type:
        "FOR_TEMPLE", direction: OUT)
        }
```

Output

Ticket ID	Date	Seat Number	Status
TK1	2025-10-01	A10	Booked

Ticket ID	Date	Seat Number	Status
TK1	2025-10-01	A10	Booked

Ticket ID	Date	Seat Number	Status
TK1	2025-10-01	A10	Cancelled

Ticket ID	Date	Seat Number	Status
(none)			

Create Nodes

```
CREATE (t:Temple {templeID: 'T1', name: 'Sacred
        Temple', location: 'CityA'});
CREATE (u:User {UserID: 'U1', name: 'Alice', email:
        'alice@example.com'});
CREATE (tk:Ticket {ticketID: 'TK1', date: '2025-10-
        01',
        seatNumber: 'A10', status: 'Booked'});
```

Create Relationship

```
Match (u:User {UserID: 'U1'}), (tk:Ticket {ticketID:
        'TK1'}), (t:Temple {templeID: 'T1'})
CREATE (u)-[:BOOKED_FOR]->(tk);
CREATE (tk)-[:FOR_TEMPLE]->(t);
```

Query Tickets for temple

```
MATCH (t:Temple {templeID: 'T1'})<-[:FOR_TEMPLE]
      -(tk:Ticket)
RETURN tk.ticketID, tk.date, tk.seatNumber,
       tk.status;
```

Find Tickets for User

```
MATCH (u:User {userID: 'U1'})-[:BOOKED_FOR]->
      (tk:Ticket)
RETURN tk.ticketID, tk.date, tk.seatNumber,
       tk.status;
```

Update Ticket status

```
MATCH (tk:Ticket {TicketID: 'TK1'})
SET tk.status = 'Cancelled'
RETURN tk;
```

Delete Ticket

```
MATCH (tk: Ticket {ticketID: 'TK1'})
DETACH DELETE tk;
```

VEL TECH - CSE	
EX NO	09
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	18
SIGN WITH DATE	

18/9/25

Result

Thus the program to design the graph space and CRUD operations is verified and successfully Completed.

25/9/25 Task 10

Normalizing databases using functional dependencies up to Third Normal form

Aim:

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

Algorithm

step 1: Unnormalized relation

step 2: First Normal form (1NF)
- Remove repeating groups.
- Ensure each field holds atomic values.
- Assign a primary key.

step 3: Second Normal form (2NF)
- Remove partial dependency
- split non-key attributes fully dependent on primary key.

step 4: Eliminate transitive dependencies.
Ensure non-key attributes depend only on primary key.

Output

Table Name	Key	Description
Devotee	Devotee ID	stores devotee person details
Temple	Temple ID	stores temple information
Pooja ID	Pooja ID	stores pooja or service types
Booking	Booking ID	stores booking details linking Devotee,
Payment	Payment ID	stores payment transaction.

Example: if payment mode details are required move them into seperate table.

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

25/9/24

Result:

Thus, the 9 relation for Online Temple Ticket Booking management system has been normalized.

9/10/25 Task 11

Aim:

To develop an online temple ticket Booking Managment application using Oracle form; that enable devotees to book ticket, manage bookings and generate summary reports using Oracle forms, Menus and Report Builder.

Algorithm

step 1- Login/Register: User logs in or registers via Oracle form.

step 2- Booking: user selects temple, pooja, date, and ticket quantity, system calculates total and process payment.

step 3- Manage Booking: user/admin can view, update, or cancel bookings via Forms.

step 4- Generate Report: Admin uses menu to generate ticket sale and visit report via Report Builder.

Output

A fully functional Oracle UI with.

- Interactive ticket booking form
- Menu-driven navigation
- Automated sales and visit reports.

VEL TECH - CSE	
EX NO	1
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	04
RECORD (5)	5
TOTAL (20)	19
SIGN WITH DATE	

Result

User can securely book online ticket online. Admins efficiently view, manage and report ~~on~~ bookings and temple visits.

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

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

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

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

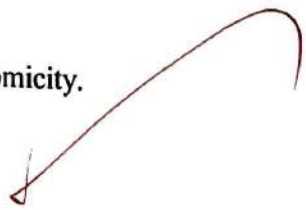
Proposed System

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.



- 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.)

ementation

ystem implementation consists of:

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

ase Tables (Normalized up to 3NF):

Table

```
E TABLE Users (
r_ID INT PRIMARY KEY AUTO_INCREMENT,
e VARCHAR(100),
il VARCHAR(100) UNIQUE,
sword VARCHAR(100)
```

ons Table

```
E TABLE Coupons (
pon_ID INT PRIMARY KEY AUTO_INCREMENT,
e VARCHAR(50) UNIQUE,
count DECIMAL(5,2),
iry_Date DATE,
tus VARCHAR(20)
```

Table

```
TABLE Offers (
r_ID INT PRIMARY KEY AUTO_INCREMENT,
ription VARCHAR(255),
d_From DATE,
d_To DATE
```

nts Table

```
TABLE Payments (
ent_ID INT PRIMARY KEY AUTO_INCREMENT,
_ID INT,
it DECIMAL(10,2),
DATETIME,
s VARCHAR(20),
GN KEY (User_ID) REFERENCES Users(User_ID)
```

ponUsage Table

```
TE TABLE CouponUsage (
age_ID INT PRIMARY KEY AUTO_INCREMENT,
er_ID INT,
upon_ID INT,
yment_ID INT,
age_Date DATETIME,
REIGN KEY (User_ID) REFERENCES Users(User_ID),
REIGN KEY (Coupon_ID) REFERENCES Coupons(Coupon_ID),
REIGN KEY (Payment_ID) REFERENCES Payments(Payment_ID)
```

---

le (Sample Implementation in Python + MySQL)

```
rt mysql.connector
datetime import datetime

tabase connection
= mysql.connector.connect(
ost="localhost",
ser="root",
password="",
database="gift_coupon_db"

or = conn.cursor()

deem coupon function
redeem_coupon(user_id, coupon_code, amount):
try:
    conn.start_transaction()

    # Check coupon validity
    cursor.execute("SELECT Coupon_ID, Discount, Status FROM Coupons WHERE
%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)
(%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,
_ID, Usage_Date) VALUES (%s, %s, %s, %s)",
                   (user_id, coupon_id, payment_id, datetime.now()))
```

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

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

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

ple usage
_coupon(1, "OFFER2025", 500)
```

---

sion

t Coupon Application effectively demonstrates how **relational databases** ensure tion **integrity** and **data consistency** for coupon and payment management systems. wing **3NF normalization** and using **ACID-compliant transactions**, the system s duplicate coupon redemption and payment mismatches.
ject successfully shows that **horizontal scalability with clustering** and **relational ization** can coexist, making it suitable for large-scale enterprise deployments.

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