Task 1: Conceptual Design after FTR (Full text Review) using basic Data 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 relation ships, cardinality type of relationship

1.d) Reframing the relations with keys and constraints.

24-7-25

Online Temple Ticket Booking Management system

Task-1: Conceptual Design after FTR (Full text review)

A online temple ticket booking management system allows the devotees to book tickets for temple visits, special darshans, poojas and other events online, reducing physical queues and improving crowd management. These system often include features like date and time slot selection, payament processing and the generation of tickets or passes.

Entity:

A real world object or concept that can be distinctly as identified.

Examples include students, courses or products.

Entity set:

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

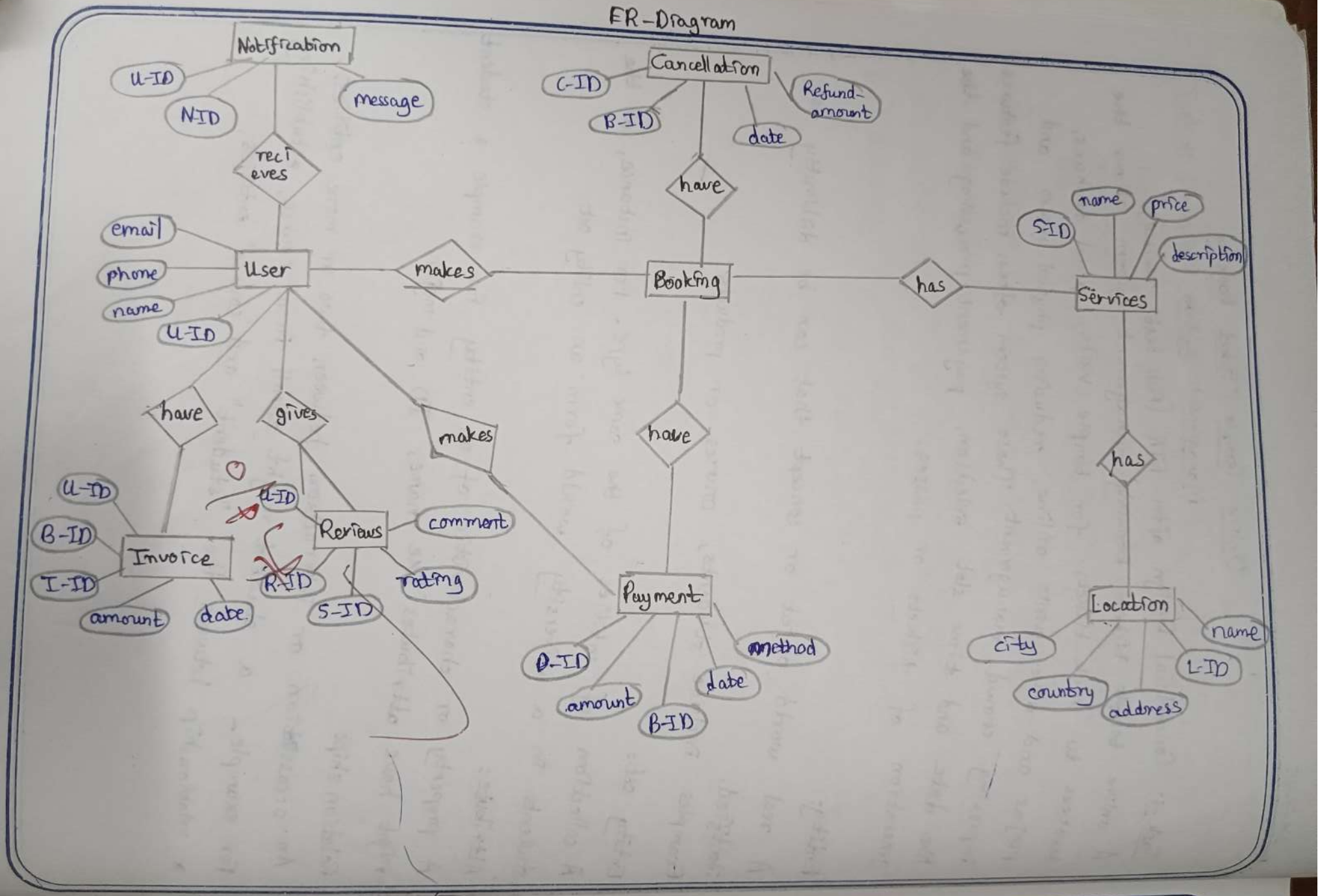
Attribute:

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

Relation ship:

An occassation or interaction between two or more entities. For example- a student might enroll in a course, establishing a relationship between the "student" and "course" entities.

ER-Diagram


Notification
U-ID
N-ID
message
receives
User
email
phone
name
U-ID
makes
Booking
Cancellation
C-ID
B-ID
Refund-amount
date
have
has
Services
S-ID
name
price
description
have
gives
makes
Invoice
U-ID
B-ID
I-ID
amount
date
Reviews
U-ID
R-ID
S-ID
comment
rating
have
Payment
P-ID
amount
B-ID
date
method
has
Location
city
country
address
L-ID
name

Aim:-

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

Attributes:

- Devotes :- Devote - 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.
- Payment :- Payment - ID, Amount, method, status, Date.
- Ticket :- Ticket - ID, issue - date, Qr - code, validity
- Admin :- Admin - ID, Name, user name, password, Race.

Types of Attributes:-

- Simple : Name, price.
- Composite : Address (street, city, state)
- Multi - valued : Phone
- Derived Age (from DOB)

Relationships:-

- Devote → 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 : Devote → 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)	3+5=18
SIGN WITH DATE	

24/7/25

Result: The ER diagram for the temple ticket booking management system was successfully designed, showing all entities, attributes and relationship with correct coordinalities for database implementation.

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 forms of inheritance.

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

2.b) Check is a high hierarchical / has a hierarchy and perform generalization and/or specilization relationship.

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

2.d) Rename the relations.

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

Generating Design of other traditional database model.

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

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

Relationship 1: Temple manages Event (one-to-many)

- Specificity: One Temple can organize many Events, but each event is managed by only one Temple.
- Surplus Relation: No surplus relation needed since it is already one-to-many

Relationship 2: Event has ticket (one-to-many)

- Specificity: Each event can have many ticket types (e.g General, VIP), but each ticket belongs to one event.
- Surplus Relation: No surplus relation needed; one-to-many is appropriate.

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

- Specificity: A customer can book many tickets, and a Ticket can be booked by many customers (for group bookings)
- Surplus Relation: Yes, surplus relation needed for many-to-many relationship e.g. 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 by 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 generalisation and specialization.

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 custome_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 Ex.
age (customer)	Positive integer (min 18)	CHECK (age >=18)
ticket_price	Positive decimal number	CHECK (ticket_price >0)
booking_date	Date (not in past)	CHECK (booking_date >= CURREN_DATE)
seat_number	String or number within valid range.	CHECK (seat_number BETWEEN 1 and 500)

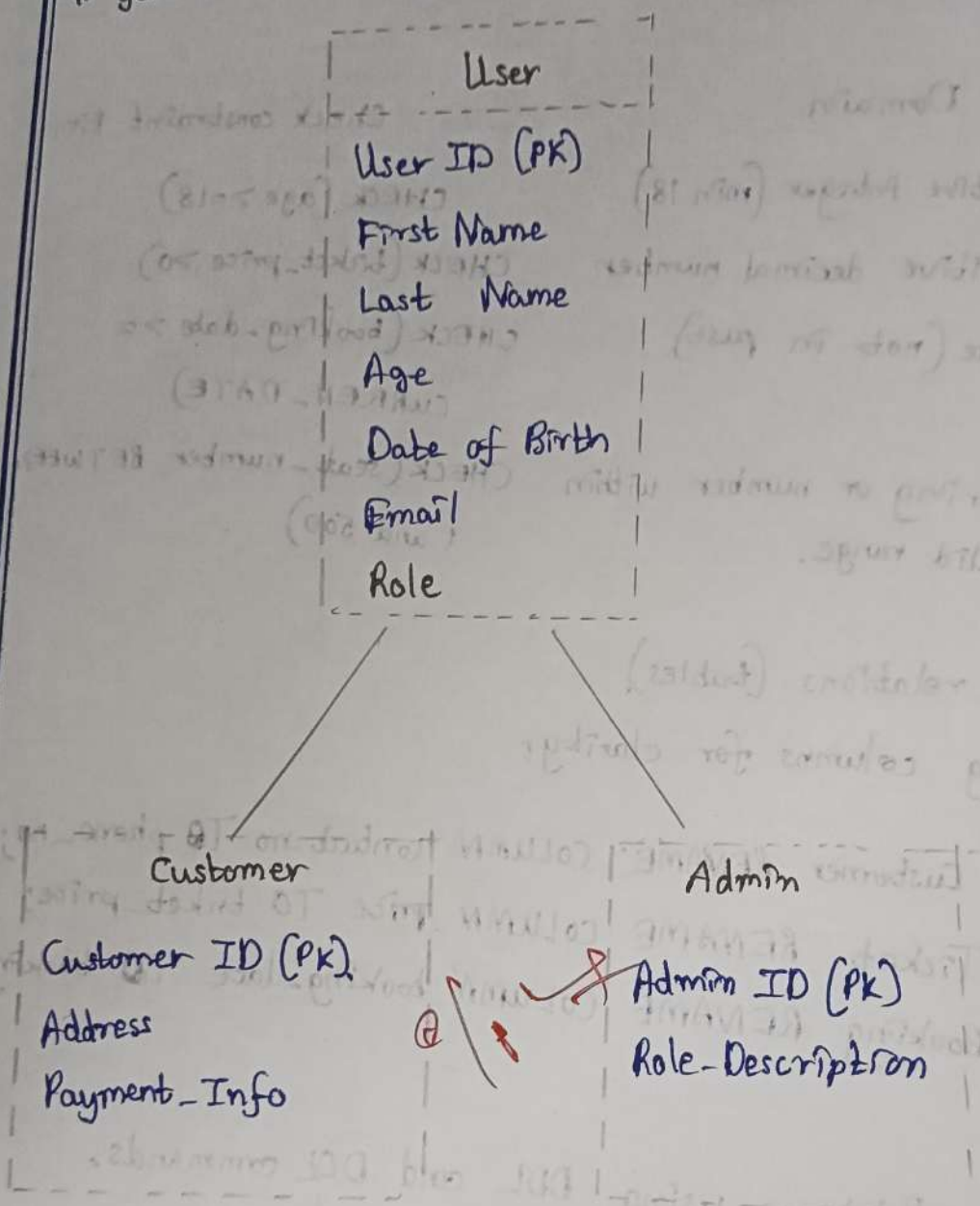
2.d Rename the relations (tables)
Example renaming columns for clarity:

```
ALTER TABLE Customer RENAME COLUMN contact_no TO phone_no;
ALTER TABLE Ticket RENAME COLUMN price TO ticket_price;
ALTER TABLE Booking RENAME COLUMN booking_date TO booking
_date;
```

2.e Perform SQL Relations using DDL and DCL commands.
DDL (Data Definition Language):
--Create Use Table (Generalization)

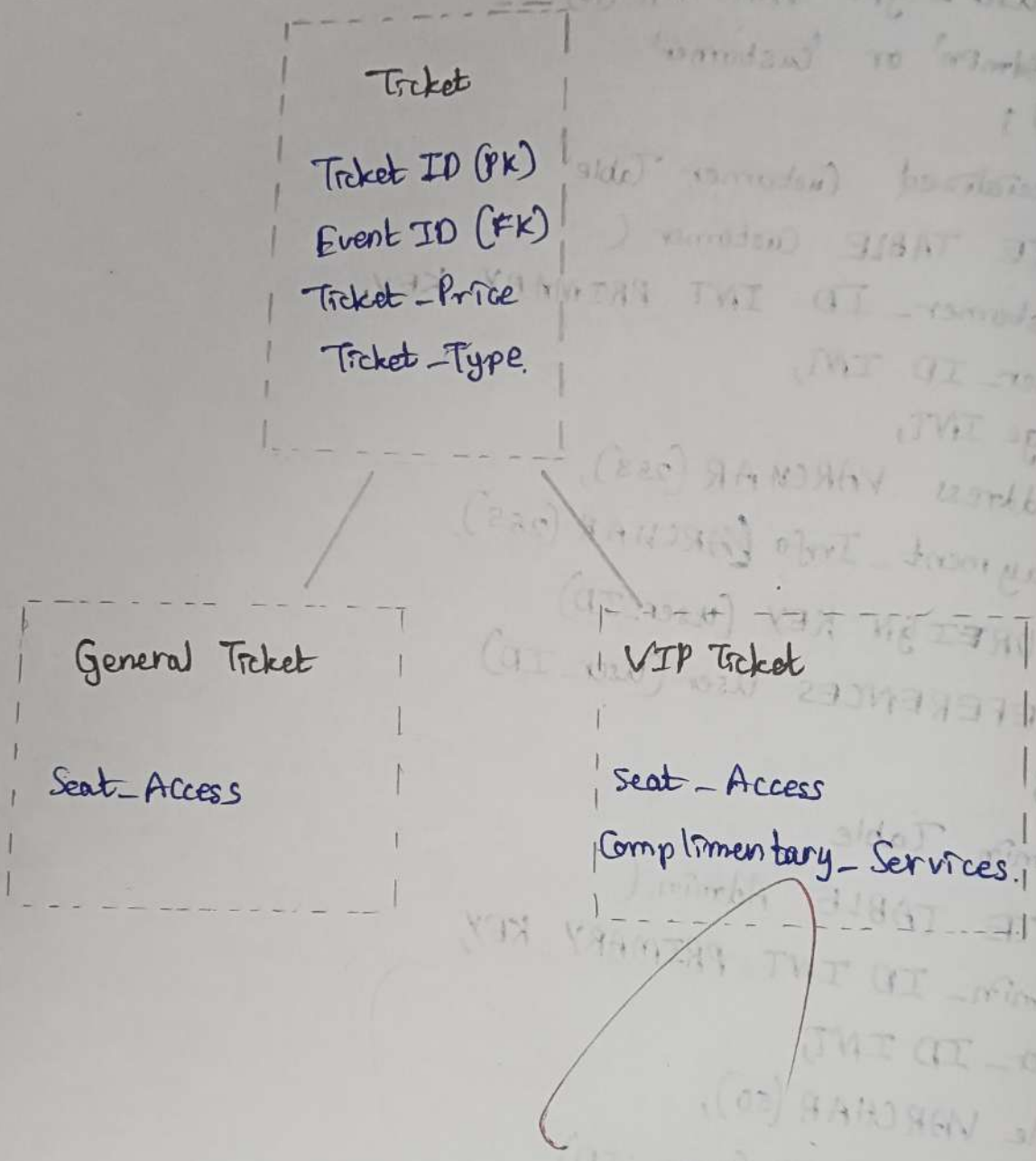
```
CREATE TABLE User (
    User_ID INT PRIMARY KEY,
    First_Name VARCHAR (50),
    Last_Name VARCHAR (50),
    Email VARCHAR (100),
```

1- Generalization Diagram: User as superclass for customer and Admin



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

2- Specialization Diagram: Ticket as Superclass for general Ticket and VIP Ticket



```
-- 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)
);
-- Ticket Table with specialization handled by ticket_type
attribute
CREATE TABLE Ticket (
    Ticket_ID INT PRIMARY KEY,
    Event_ID INT,
    Ticket_Type 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 (Customer_ID),
   FOREIGN KEY (Ticket_ID)
   REFERENCES Ticket (Ticket_ID)
);
```

VEL TECH - CSE	
EX NO	2
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	18
SIGN WITH DATE	

31/08/25

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

7/8/25

Task-3:

Using clauses, operators and functions in queries:

Aim: To perform query processing on a Temple Ticket online booking management system to different retrival results of queries using DML, DEL operations with aggregate functions, data function, string function, set clauses and operations.

Temple:

TEMPLE ID	NAME	LOCATION	CONTACT-NO
TID01	Tirumala venkateshwara	Tirupati	9014276954
TID02	Meenakshi Aman.	Madurai	9789222752
TID03	Kashivishwanath	Varanasi	7013982362
TID04	Jagannath Temple	Puri	9701224404
TID05	Golden Temple.	Amritsar	9963742315

Visitor:

Visitor_ID	F_Name	L_Name	Age	Email	Contact_No
V001	Anil	Kumar	30	Anil@gmail.com	9268453212
V002	Akash	Reddy	25	Akashr@gmail.com	9701224404
V003	Priya	sharma	28	Priyas@gmail.com	9905290010
V004	kumari	Priya	35	kumari@gmail.com	7013982716
V005	Swetha	Rao	22	Swethar@gmail.com	7989222725

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
B003	TID03	V003	2024-06-17	General	100
B004	TID04	V004	2024-06-18	VIP	700
B005	TID05	V005	2024-06-19	General	200

Priest:

Priest ID	F_Name	L_Name	Age	Email	Contact_No
P001	Ramesh	Iyer	50	ramesh@gmail.com	9044872652
P002	Suresh	Sharma	45	suresh@gmail.com	9827621251
P003	Leo	Das	40	manish@gmail.com	8654327191

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

```
SELECT *
FROM Visitor
WHERE F_NAME LIKE 'A%';
```

Result:

Visitor ID	F_Name	L_Name	Age	Contact No.
V002	Akash	Reddy	25	9701224404

3. Add a column for 'special-sera', in Booking table.

```
ALTER TABLE Booking ADD Special Bava VARCHAR(50);
```

Result:

Table altered successfully

4. Count the number of VIP ticket bookings.

```
SELECT COUNT(*)
FROM Booking
Where Ticket_type = "VIP";
```

Result:

Count(*)

5. Display temple details for Temple 'RDS' 'TIDO1', 'TIDO3' & 'TIDO4'

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

Result:

Temple ID	Name	Location	Contact_No
TIDO1	Tirumala	Tirupati	9701224404
TIDO2	Kashi	Varanasi	70-13952716
TIDO3	Golden temple	Amritsar	7989222725

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

```
SELECT visitor ID, F NAME, L_NAME
FROM Visitor
WHERE Visitor ID IN(
```

SELECT Visitor ID FROM Booking WHERE Ticket_type = 'special'
);

Result:

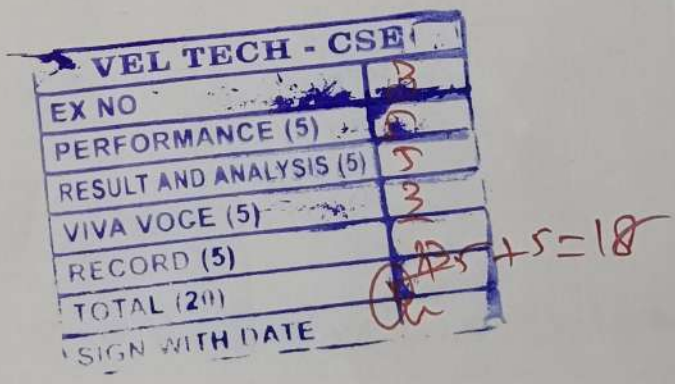
Visitor ID	F_NAME	L_NAME
V005	Sneha	Rao.

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

SELECT Priest ID
FROM Priest
WHERE TEMPLE ID IS NULL

Result:

Priest ID
(No Result if all priests are assigned)



Result: Thus, queires 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 corrected and nested Sub-queries such as finding summary statics in the temple ticket online booking management system:

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

```
SELECT  t.Temple ID,
        t.Name As Temple Name,
        t.location,
        t.Contact_ID,
    COUNT (b.Booking ID) As Total Booking
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	Temple Name	Location	Total Booking.
TID01	Tirumala	Tirupati	3
TID02	Meenakshi Amman	Madhurai	1
TID03	Kashi	Varanasi	1

TID04	Jagannadh temple	Puri	0
TID05	Golden temple	Amritsar	0.

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

```
SELECT t.Name As Temple Name,
COUNT (*) As Total Special Bookings
FROM Temple t
Join Temple b.
ON t.Temple ID = b.Temple ID
WHERE b.Ticket_type = 'special'.
Group By t.Name;
```

Output:

Temple Name,	Total special Booking.
Tirumal venkateshwara	

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

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

Output:

Temple ID	Name	Location	Contact_No.
TID01	Tirumala	Tirupati	9701224404
TID02	Kashi	Varanasi	7013982716

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

```
SELECT V.Visitor ID
       V.F Name AS Visitor Name,
       b.Booking ID,
       b.Booking_Date,
       b.Ticket_type,
       b.Amount.
FROM Visitor
JOIN Booking b
ON V.Visitor ID = b.Visitor ID
WHERE V.Age > 25;
```

Output:

Visitor ID	Visitor Name	Age	Booking ID	Amount
V001	Anil	30	B001	500
V003	Priya	25	B003	100
V004	Swetha	35	B004	700

4.5: To retreive the details of Temples with no bookings.

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

Output:

Temple ID	Name	Location	Contacts_No.
TID04	Jagannad temple	Puri	9701224404
TID05	Golden temple	Amritsar	9014276954

4.6: To retreive the temple id, temple name, and visitor name for a partticular visitored given.

```
SELECT t.Temple ID,
       t.Name As Temple Name,
       V.FNames as Visitor Name
FROM Temple t
Join Booking b
ON t.temple ID = b.Temple ID.
Join visitor V
 ON b.visitor ID = V.visitor ID
 WHERE V.visitor ID = 'V005';
```

Output

Temple ID	Temple Name	Visitor Name.
TIDD1	Tirumala	Sneha
TIDD2	Meenakshi Amman	Swetha

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

14/8

Result: Thus, the queries using function joins and nested subqueries were successfully executed for the temple ticket online booking management system.

21/8/25

Task-5

Writing Join Queries, Equivalent, and/or recursive Queries :

(Tool : SQL/Oracle, Aim: Flipped Classroom).

Aim: To perform advanced query process and best its heuristics using join queries, equivalent queries, and recursive queries for the temple ticket online booking management system, which manages devotes, priests, poojas, tickets and booking details.

Queries and Outputs

5.1: To retrieve all temples and their poojas.

```
SELECT E.Name As temple, A.Pooja Name, P. Price
FROM Temple T
Join pooja P.No - E. Temple ID = P. Temple ID;
```

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

```
SELECT b. Booking ID, d. Name As devotee, P. Pooja name,
b. Booking Date, b. slot time, b. status.
FROM Booking b.
JOIN Devote ON b. Devotes & IP = d. Devotes ID
JOIN Pooja P ON b. Pooja ID = P. Pooja ID.
```

Output:

Temple	Pooja Name
Tirupati Temple	Suprabhaba Seva
Tirupati Temple	Archana
Kanchi Temple	Abhishekam
Madurai Temple	Special Darshanam

5.3: Count the number of booking made by each devotee.

```
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 d
Group By d.FName;
```

5.4: To find out devotes who booked 'Abhishekam'

```
SELECT d.devotes ID, d.F Name, d. Mobile No,
FROM Devote d.
JOIN Booking b on d.devote ID = b.Devote ID
JOIN Pooja p on b.Pooja ID = P.Pooja ID.
WHERE P.Pooja Name = 'Abhishekam';
```

Output :

Booking ID	Devote	Pooja Name	Booking Date.
B101	Rajesh	Suprabhata seva	22-Jun-2023
B102	Meena	Archana	22-Jun-2023
B103	Arjun	Abhishekam	23-Jun-2023
B104	Jagan	Special darshan	23-Jun-2023
B105	Bharathi	Abhishekam	25-Jun-2025

Output:

Devote	Total Bookings
Rajesh	1
Meena	1
Arjun	1
Jagan	1
Bharathi	1

5.5: To retreive all pooja and the number of times that were booked.

```
SELECT P Pooja Name, COUNT (b.booking ID/ON)
Total Bookings
FROM Pooja.P
LEFT Join Booking b ON P. Pooja ID = b.Pooja ID
GROUP By P. Pooja Name;
```

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

```
SELECT t.Name as Temple, Count (b.Booking ID).
As cancelled bookings
FROM Temple T
JOIN Booking b ON T. Temple ID = b. temple ID
WHERE b status = "cancelled".
GROUP By T. Name;
```

5.7: To Retrieve temple details where booking was successfull.

```
SELECT DISTINCT t. Temple ID, t. Name, t. Location,
t. Head Priest.
FROM Temple t
JOIN Booking b on t. Temple ID = b. Temple ID
WHERE b.status = 'comfirmed'
```

Output:

Devotee ID	F Name
D103	Arjun
D105	Bharathi

Output:

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

Output:

Temple	Cancelled Booking
Kashi Temple	1

5.8: Retrieve devotee and booking details for devote above 50 years old.

```
SELECT d.F_Name, d.Age, b.Booking ID, b.Booking Date,
 b.Slot time, b.Status.
FROM Devote d
JOIN Booking b ON Devoted ID = b.Devote d.ID
 WHERE d.age >50;
```

5.9: To Retreive the details of temple where no Pooja booking all done.

```
SELECT
FROM Temple
 WHERE Temple ID Not (SELECT Temple ID From Booking);
```

5.10: To Retreive temple, pooja and devote d details for a given Booking ID.

```
SELECT t.Name As Temple, P. pooja Name, d. F Name
 As Devote,
      b.Booking Date, b. Slot time.
FROM Booking b
JOIN Temple t ON b. Temple ID = t.temple ID
JOIN Pooja P ON b. Pooja ID = P.pooja ID
JOIN Devote d ON b.devobe ID = d.Devobe ID
 WHERE b.booking ID = 'B102';
```

Output:

Temple ID	Name	Location	Head Priest
T001	Tirupati	Tirupati	Ramanujan
T002	Kanchi	Kachipuram	Narayana Das
T003	Madurai	Madurai	Subramanyam

Output:

F Name	Age	Booking ID	Booking Date	Slottime	Status
Suresh	55	B105	24-Jun-2023	6:30	Confirmed

Output:

Temple ID	Name	Location	Head Priest.
T001	Rameshwaram Temple	Rameshwaram	Vishnu Devan

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

24/8/25

Result: Thus the temple ticket online booking management system queries using Join Queries, Equivalent Queries, and Recursive Queries was executed successfully to manage temples, devotees, priests, pooja, and booking

28/8/25

Task-6

Procedures, Functions, and Loops using PL/SQL on Number Theory and Business Scenarios.

Aim: To write and execute PL/SQL procedures, functions and loops on Number Theory (e.g. Prime Number, Factorial) and Business scenario (e.g. Salary calculation, Discount Calculation).

SOFTWARE USED:

Oracle SQL/PL/SQL

A) Number Theory Example - Prime Number Checking using procedure

1. Start
2. Initialize counter
3. Read a number n
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 < 2000 → 20% bonus.

If salary b/w 2000 - 50000 → 10% bonus

Else → 5% bonus.

4. Return final salary with bonus
5. End.

Program:

(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;
     END IF;
    END LOOP;
  IF flag THEN
     DBMS_OUTPUT.PUT_LINE (n || ' is Prime.');
  ELSE
     DBMS_OUTPUT.PUT_LINE (n || ' is NOT Prime.');
  END IF;
  END IF;
END;
/
--Execution
BEGIN
   check_prime(29);
   check_prime(20);
END;
/
```

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 ;
   ELSIF salary BETWEEN 20000 AND 50000 THEN
      bonus := salary * 0.10;
   ELSE
      bonus := salary * 0.05;
   END IF;
   RETURN (salary + bonus);
END;
/
-- Execution
DECLARE
   final_sal NUMBER;
BEGIN
   final_sal := calc_bonus (18000);
   DBMS_OUTPUT.PUT_LINE('Final Salary with Bonus: ' || final_sal);
   final_sal := calc_bonus (30000);
   DBMS_OUTPUT.PUT_LINE ('Final salary with Bonus: ' || final_sal);
END;
/
---
```

Output:

For Prime Number Check:

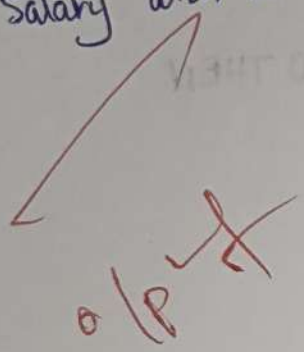
29 is Prime.

20 is NOT Prime.

For Bonus Calculation:

Final Salary with Bonus: 21600

Final Salary with Bonus: 33000



VEL TECH - CSE	
EX NO	6
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	15+5=20
SIGN WITH DATE	

20/8/25

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

4/9/25

Task-7

Triggers, Views and Exceptions.

Aim: To conduct events, views and exception handling on CRUD operations for the Temple Ticket Online Booking Management 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 ROW
BEGIN
    INSERT INTO Ticket (Ticket ID, Booking ID, Devotee Name, Age,
                        Gender, QR Code, Status).

    VALUES (
     set_ticket.NEXTVAL,
sequence for ticket IDs.
    :NEW.Booking ID,
    'Unknown',
placeholder
      NULL,
given at booking time
      NULL,
given at booking time
      SYS_GUID(),
```

```
auto_generate QR code
        'ACTIVE'
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.BookingRef,
    u.Name AS Devotee
    t.Name AS Temple
     dt.Name AS Darshan Type,
     s.StartsTs AS slot Start,
     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.Temple ID = t.temple ID
 JOIN slot s ON b.Slot ID = s.Slot ID
 JOIN DarshanType dt ON s.Darshan Type ID = dt.Darshan Type ID;
```

c) Non-Recursive PL/SQL Procedure:

Requirement:

Retrieve even-numbered Booking IDs for any given temple (similar to even Playe IDs).

SQL:

```
CREATE OR REPLACE PROCEDURE
Get Even Booking IDs For Temple (
    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;
/
```

Output:

Booking ID	Booking Ref	Devote	Temple	darshan type	Slot start	Slot end		Amount
101	BK202 3001	Ramesh	Tirumala	General	25-11-23	6:00	7:00	200
102	BK202 3002	Priya	Priya	VIP	13-11-23	8:00	9:00	500
103	BK202 3003	Amibe	Amibe	special	17-11-23	10:00	11:00	800

Execution Block:

SQL:

```
DECLARE
    temple_id NUMBER := 101;
    even_booking_ids SYS.ODCINUMBERLIST;
BEGIN
    Get Even Booking IDs For Temple (temple_id, even_booking_ids);
FOR i IN 1.. even_booking_ids.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE ('Even
Booking ID: ' || even_booking_ids (i));
    END LOOP;
END;
/
```

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

Result: Thus, the Triggers, Views and Exceptions for the Temple Online Ticket booking Management System were successfully implemented and verified.

11/9/25

Task-8

CRUD operations in document database

Aim: To design a document database for online temple ticket booking management system using mongoose (NPM) with mango DB and perform CRUD operations: Create, Read, update, delete tickets, temples, and the user booking

Algorithm:

1. Install Mango DB community Edition
2. Install Mongo shell (Mongah)
3. Configure path environment variable to include mongosh binary
4. Confirm Installation
   mongosh --help
5. Start Mongo DB server
   C:\Program Files\Mongo DB\server\bin\Mongod.exe
6. Perform CRUD operations on Temple online ticket booking database.

Collection : temples.

```
{
  "Temple ID": "TI001",
  "Name": Meenakshi Amman Temple",
  "Location": "Madurai",
  "contact": 9876543210",
  "Timings": "6:00Am - 9:00 PM".
```

Output:

```
{
  "acknowledged": True;
  "Inserted ID";
  Objected ID("651 d1C9E6bbfe 793ad8910 11)
}
```

Output:

```
{
  "acknowledged": True;
  "Inserted ID"
  object ID ("651d1ea06ebbfe 7993ad913")
}
```

Output:

```
{
  "_id":
  object id ("651d1ee06ebb8c7993ad913");
  "Ticket id": 'TIKD01",
  "Temple id": 'TIDO1",
  "User name": "John Doe",
  "Booking Date": 2025-09-10",
  "Number of persons": 3
}
```

Collection: tickets

```
{
 "Ticket ID": "TkID01",
 "Temple ID": "TID01",
 "Username": "John Doe",
 "Booking Date": "2025-09-10",
 "Number of persons: 3'.
}
```

CRUD operations

1) Create collection.

```
db.create collection ("Temples");
db.create collection ("Tickets");
```

2) Insert documents.

→ Insert temples

```
db.temple.insert one ({
        Temple ID: "TID01",
        Name : "Meenakshi Amman temple",
        Location: "Madurai",
        Contact : 9876543210,
        Timings: "6:00Am-9:00PM"
    });
db.temples.Insert many ([
  {Temple ID: "TID02", Name: "Ramanath swamy Temple"
    Location: "Rameswaram", Contact: 9876543210, Timings:
    "5:00AM-8:00PM"},
```

{ Temple ID: "TID03", Name: "Brihadeswar temple",
Location: "Thanjavur", Contact: 9687253214, Timings:
"7:00 Am - 8:30 PM" }]);

→ Insert ticket booking,

db. Tickets. Insert one ({
Ticket ID: "TKID01",
Temple ID: "TID01",
username: " John Doe",
Booking date: "2023-09-10",
Number of persons: 3
});

Read / Query document,

→ Find all temples.

db. temples. find (). Pretty ();

→ Find ticket by user.

db. Tickets. find ({username: "Johndoe"}). Pretty ();

4) Update documents

→ Update Temple contact number

db. temples. Update one (
{ Temple ID: "TID01"},
{ Set : { contact: 9988675432}
});

5) Delete documents

→ Remove ticket booking:

db. Tickets. delete one ({ Ticket ID: "TKID01"});

Output:

```
{
 "acknowledged": True,
 "Matched count": 1,
 "Modified count": 1,
}
```

Output:

```
{
 "acknowledgement": True,
 "deleted current": 1.
}
```

Output:

```
{
 "acknowledgement": True,
 "deleted count": 1
```

→ Remove a temple.

db.Temple.delete({Temple ID: "TID03"});

VEL TECH - CSE	
EX NO	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	2
RECORD (5)	5
TOTAL (20)	17
SIGN WITH DATE	

Result: Thus, using MongoDB with Mongoose was successfully designed a document database for an online Ticket Booking Management system and performed using CRUD operations.

18/9/25

Task-9

Crud operations in Graph database.

Aim: To perform CRUD operations like, creating, inserting, querying updating and on graph online temple ticket booking management system.

Algorithm:

1. Start on click "start Free".
2. Choose the option to "continue with google".
3. Click the "open" button.
4. After clicking "open" a text will file be automatically downloaded This file contains your user ID and password details.
5. Copy the password from the downloaded text file and paste it where required.
6. Close the "Get started with Neo4j with beginner guides" if it's open
7. You're now ready to begin practising with the Graph database.

Create Nodes with Properties:

Each node represents an entity like Devotee, temple & booking

Create a Devotee Node:

CREATE (d:Devotee{DevoteeID: 'D001', Name: 'Raj kumar', Age: 30, phone: '9876543210', Email: 'rajkumar@gmail.com')

RETURN d.

Create a Temple Node:

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

Create a Booking Node:

```
CREATE (b: Booking {Booking ID: 1
    Temple ID: 'T001', Date: '2025-09-10', No. of Tickets:})
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 bt.
```

Display All nodes.

```
Match (n) RETURN (n)
```

Retrieve Particular Booking:

```
MATCH (b: Booking {Booking ID: 'B001'})
RETURN b.
```

Output:

d-Devotee ID	d-Name	d-Age	d-phone	d-Email
D001	Raj kumar	30	9876321432	rajkumar@gmail.com

Output:

t-Temple ID	t-Name	t-Location	t-capacity
T001	Sri venkates wara temple	Tirupati	5000.

Output:

b-Booking ID.	b-DevoteeID	b-Temple ID	b-date	No. of tickets
B001	D001	T001	25-9-20	2

Output:

b. booking ID	b. Devotee ID	b Temple ID	d. date	No. of tickets
B001	D001	T001	25-09-20	2

Output

d. Devotee ID	d. Name	d. Age	d. phone	d. Email
D001	Raj Kumar	31	9123457862	rajkumar@gmail.com.

Update Particular Devotee Details.

```
MATCH (d:Devotee {DevoteeID: 'D001'})
SET d.phone = '9123478650', d.Age = 31
RETURN d.
```

Delete Particular:

```
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 D. [illegible]	

Result: 18/9

Thus, we successfully created, inserted, queried, updated, and deleted nodes and relationships in the temple online ticket booking management system using "Neo4j" Graph Database.

25/9/25

Task-10

Normalizing database using functional dependencies up to Third normal form.

Aim:

To normalize the below relation and create the simplified tables with suitable constraints for the temple online ticket booking management system.

Given relation:

Temple Booking(

Booking ID, User ID, U Name, U Email, U contact, Temple ID, T location, T name, priest ID, priest Name, priest contact, Ticket ID, booking date, Darshan date, time slot, ticket type, payment ID, payment mode, payment status, Transaction ID.

).

Step 1: Apply functional depencies → Normalise to INF

first, identify functional depencies (FDS).

1- User ID → Name, T Location.

2. Temple ID → T Name, T location

3. Priest ID → P. Name, P contact

4. Ticket ID → Ticket type, Price

5. Payment date → User ID, Temple ID, priest ID, ticket ID, Booking date, Darshan date, Time slot, payment ID,

Step 2. Normalize using FDt and At

Candidate key:

* Booking ID (main unique identifies of booking from at
* Booking ID: { autributey → confirm reciving ID is a candidate key
* User ID, Temple ID, priest ID, Ticket ID, payment ID also act as foreign key

Step - 3 Minimal cover/canonical cover.

We reduce FD s to minimal form:

1. User ID → U name, U Email, U contact
2. Temple ID → T name, T location
3. Ticket ID → Ticket type, price.
4. Priest ID → priest name, priest contact.
5. Payment ID → payment mode, payment status.
6. Booking ID → User ID, Temple ID, Priest ID, Ticket ID, Booking Date, Darshan Date, Time slot, payment ID.

Step 4: Normalize To 2NF.

Conditions for 2NF:

- Must already be 1NF
- No partial dependency

Here Booking ID is a primary key.

Step 5: Normalize to 3NF.

conditions for 3NF:

- Must already be 2NF
- No partial transitive dependencies.

check:

- Booking ID → User ID → U Name.
- Booking ID → Temple ID + name,
- Booking ID → priest ID + priest name
- Booking ID → Ticket ID + Ticket type, price.
- Booking ID → payment ID + payment status.
- ID resolve → separate no different relations.

Final simplified tables in 3NF.

1. User Table.

User (User ID, U name, U Email, U contact)

2. Temple table.

Temple (Temple ID, T name, T location)

3. Priest table.

Priest (priest ID, priest name, contact).

4. Ticket table

Ticket (Ticket (pk), Ticket type, price)

5. Payment table

payment (payment ID (pk), payment mode, payment status, transaction date)

6. Booking table.

Booking (BookingID (u), Temple ID (pk), priest ID, Ticket ID (fk))

Constraints:

- Primary key: Booking ID, Temple ID, priest ID
- Foreign key:
  - Booking user ID → User, User ID.
  - Booking Temple ID → Temple, Temple ID
  - Booking, priest ID → Ticket, Ticket ID.
  - Booking, Payment ID → payment, payment ID.

Unique constraints: U Email, U contact.

Check constraints.

price = 0

payment shift ['pending', complete, failed]

. Time slot within 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	

25/9/25

Result: Thus, the given reason for the online temple, booking management has been normalized into simple tables to 3NF with suitable constraints, reducing redundancy and improving data integrity.

9/10/25

Task-11

Design an online ticket booking management system using oracle forms, menus, report Builder.

Aim: To design an online temple ticket booking management system using oracle forms, menus and report Builder.

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 the database schema for the temple ticket booking system, use.

- Temples
- Ticket types
- Booking Details
- User ID Devotees
- Payment Information.

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

Create Menu:

Menu provides the navigation structure for the booking system application.

Steps to create menu in oracle Forms:

1- Open oracle forms builder.

2. Create a new menus from or use an existing one

3. Add menu items for each major function;

- View Booking history
- Cancle tickets
- Manage temples
- Reports.

4. Define menu hierach
5. Assign triggers or procedures to handles menu.
6. Compile and run the new menu from to test the navigation flow.

Design Forms:

Form's are used to capture, display and edit data related to the ticket booking.

Steps to design:

1- Create a new form for each major component.
- Ticket booking
- User registration.
- Payment processing
- Booking cancellation form.

2. Add form elements like text files (for user info), buttons (submit / cancel).
3. Use the property to configure element properties and link data blocks to data base tables.
4. Write PL/SQL code for bussines logic such as:
- Validating booking dates.
- Checking tickets availability.
- Processing payments.

Create Reports:

Reports provide summarized information about booking and temple visits.

Step's to create:

1- Open oracle Report builder

2- Create a new report or use an existing template

3: Define the data structure source for the report using queries or PL/SQL procedures.

reports:

- Daily ticket sales.
- Booking summary by temple.
- Revenue report.

4. Design a report layout with headers, and detailed data columns

5. Add parameters to allow retriving data, range, temple Name).

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

VEL TECH - CSE	
EX NO	11
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	3
TOTAL (20)	19
SIGN WITH DATE	

04/12/25

Result: The system is successfully delivered to facilitate online ticket booking and efficient temple management.

Use Case 3: A gift coupon application that handles offers and payment-related information.

Choose the database system, a relational database, for its capability to scale horizontally while keeping the ACID property. The last was particularly important because transactional data was being handled. Eventual consistency as offered by other databases was not suitable. Going with DB and cluster gives the opportunity to scale horizontally for a large number of writes and reads without compromising the ACID and transactional capability required by the application. Will it transact and lead to no deadlock, keeping all relational tables normalized? If so what normal form do they sustain to offer a gift coupon application?

Team Members:

NEELISETTY VENKATA SAI THARUN

SUNKESWARA SUSHANTH

SHAIK GOUSE MOHIDDIN

DODLA SANTHOSH

MANNURU MEDHINI

KALLURI PEDDA BABU

CHANDRAJIT KAMALAKKANNAN

VADAMADHURA DILEEP KUMAR

YARASANI TARUN KUMAR REDDY

KOMMINA SRUTHI

GOSU BHARGAV RAM

SHAIK SHAHIL ROHAN

THONDAPU SAI JASWANTH

DASARI KOTISURYA

RAVURI NAGA DURGA PRASAD

CHAMARTHI TEJESWAR RAJU

NAGANABOYINA MOHAN VENKAT

KANUPARTHI POOJITH REDDY

KONDAPATURI MADHU SUMANTH

KUCHANA SRAVIKA

VEPURALA VIPIN RAJ

GANAPATHIRAJU JAGANNADHA VARMA

KODURU KESAVA

GANDLURU VENKATA ARAVIND

ANISETTI ROHITHA

AIM:

Build a backend component that:

1. Stores coupons/offers and payment records.
2. Validates & applies coupons to orders (checks expiry, usage limits, min-order, allowed users).
3. Calculates final payable amount and records the payment + coupon usage.

ALGORITHM (high level)

1. Receive order request: (user_id, cart_amount, coupon_code, payment_method).
2. Load coupon by code. If not found → reject.
3. Validate coupon:
   - check active flag
   - check current_date ≤ expiry_date
   - check total_usage and per_user_usage limits
   - check min_order_amount
   - check if user is allowed (optional)
4. Compute discount:
   - if type = percentage → discount = min(cart_amount * pct/100, max_discount)
   - if type = fixed → discount = fixed_amount
5. Final_amount = max(cart_amount - discount, 0) + taxes/shipping (if any).
6. Process payment via gateway (simulate). If success:
   - create payment record (status = success)
   - increment coupon usage counters
   - return success + invoice

   Else:
   - create payment record (failed)
   - return failure reason

CREATE TABLE users (

```
CREATE TABLE coupons (
 id SERIAL PRIMARY KEY,
 code VARCHAR(50) UNIQUE,
 type VARCHAR(10),          -- 'percentage' or 'fixed'
 value NUMERIC,             -- percent or fixed amount
 max_discount NUMERIC NULL,
 min_order NUMERIC DEFAULT 0,
 expiry DATE,
 active BOOLEAN DEFAULT TRUE,
 usage_limit INT DEFAULT 0,   -- 0 = unlimited
 per_user_limit INT DEFAULT 1
);

CREATE TABLE coupon_usage (
 id SERIAL PRIMARY KEY,
 coupon_id INT REFERENCES coupons(id),
 user_id INT REFERENCES users(id),
 used_count INT DEFAULT 0
);

CREATE TABLE payments (
 id SERIAL PRIMARY KEY,
 user_id INT REFERENCES users(id),
 coupon_id INT NULL REFERENCES coupons(id),
 amount NUMERIC,
 discount NUMERIC,
 final  amount NUMERIC,
```

1.);

Find coupon:

```
SELECT * FROM coupons WHERE code = 'WELCOME10';
```

2. Check total usage:

```
SELECT COALESCE(SUM(used_count),0) AS total_used
FROM coupon_usage WHERE coupon_id = 123;
```

3. Get user's usage:

```
SELECT used_count FROM coupon_usage WHERE coupon_id = 123 AND user_id = 45;
```

4. Record successful payment:

```
INSERT INTO payments (user_id, coupon_id, amount, discount, final_amount, status, method)
VALUES (45, 123, 1000, 100, 900, 'success', 'card');
```

5. Increment user coupon usage (insert or update):

6. INSERT INTO coupon_usage (coupon_id, user_id, used_count)
7. VALUES (123,45,1)
8. ON CONFLICT (coupon_id, user_id) DO UPDATE
9. SET used_count = coupon_usage.used_count + 1;

10. EXAMPLE: Concrete Coupon + Walkthrough

11. Coupon row:

id	code	type	value	max_discount	min_order	expiry	usage_limit	per_user_limit
123	WELCOME10	percentage	10.00	150.00	500.00	2025-12-31	1000	1

12. User places order:

user_id = 45

- cart_amount = 1200.00
- coupon_code = WELCOME10

Validation:

- cart >= min_order (1200 >= 500) → OK

Assume tax 18% on final_amount (optional):

- tax = 1080 * 0.18 = 194.40
- payable = 1080 + 194.40 = 1274.40

Payment simulation success.

SAMPLE PROGRAM OUTPUT(Console-style)

Applying coupon: WELCOME10

User: 45 Cart: 1200.00

Coupon valid — percentage 10% (max 150.00)

Discount applied: 120.00

Amount after discount: 1080.00

Tax (18%): 194.40

Final payable amount: 1274.40

Payment status: SUCCESS

Payment ID: PAY_20251017_001

Coupon usage updated for user 45 (used_count = 1)

user_id = 45

- coupon_code = WELCOME10

Validation:

- cart >= min_order (1200 >= 500) → OK
- percentage 10% → raw discount = 120.0 → below max_discount(150) → discount = 120.00
- final_amount before tax = 1200 - 120 = 1080.00

Assume tax 18% on final_amount (optional):

- tax = 1080 * 0.18 = 194.40
- payable = 1080 + 194.40 = 1274.40

Payment simulation success.

Coupon valid — percentage 10% (max 150.00)

Discount applied: 120.00

Amount after discount: 1080.00

Tax (18%): 194.40

Final payable amount: 1274.40

Payment status: SUCCESS

Payment ID: PAY_20251017_001

Coupon usage updated for user 45 (used_count = 1)

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

16/10/25

Result:

Thus the, Stores coupons/offers and payment records validates and applies coupons to order the calculates final payable amount and records completed successfully