S. No
1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
12.

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: R. N. Durga Prasad

VTU.NO: 27474

REG.NO: 24UECL0046

BRANCH: CSE CAIML

YEAR/SEM: 2nd year / 3rd sem

SLOT: S10 L12

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

: R. N. Durga Prasad. **BRANCH** : CSE (AIML)

: 27974 **REG.NO.** : 24UECL0046

M : 2nd year / IV sem **SLOT NO.** : S10 L12

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

R. N. Durga Prasad 30/10/25

URE OF LAB HANDLING FACULTY

SIGNATURE OF HOD

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

AL EXAMINER

EXTERNAL EXAMINER

INDEX

	Title	Page No.	Marks	Faculty Signature
	Conceptual Design after Formal Technical Review	1	18	24/7
	Generating design of other traditional database models	5	18	31/7
	Developing queries with DML Single-row functions and operations	10	18	7/8
	Developing queries with DML Multi-row functions and operations	14	18	14/8
	Writing Join Queries, equivalent, and/or recursive queries	18	18	21/8
	Writing PL/SQL using Procedures, Function	23	18	28/8
	Writing PL/SQL using Loops	25	18	4/9
	Normalizing databases using functional dependencies up to BCNF	29	17	11/9
	Backing up and recovery in databases	31	19	(8/9
	CRUD operations in Document databases	33	19	18/9
	CRUD operations in Graph databases	37	18	9/10
	Micro Project: UC-3 Swiggy Coupon Application.	40.	18	16/10

Completed

199

Total Marks: ~~217~~ /240

R.T.

Signature of Faculty

24/07/2025 Task-1

Q. Temple ticket online booking management system

A temple ticket online booking management system allows devotes to book tickets for temple visits. special darshan, Poojas, and other events online, reducing physical Queues and improving crowd management. These Systems other include techtaks like data 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 Studets courses or Products.

Entity set :-

A collection of Entities of the same type for instance are Students in a university would form an Entity that.

Attributes :-

A Property or characteristic of an Empty for Example, a student might have attributes like name : ID and major

Relationship :-

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

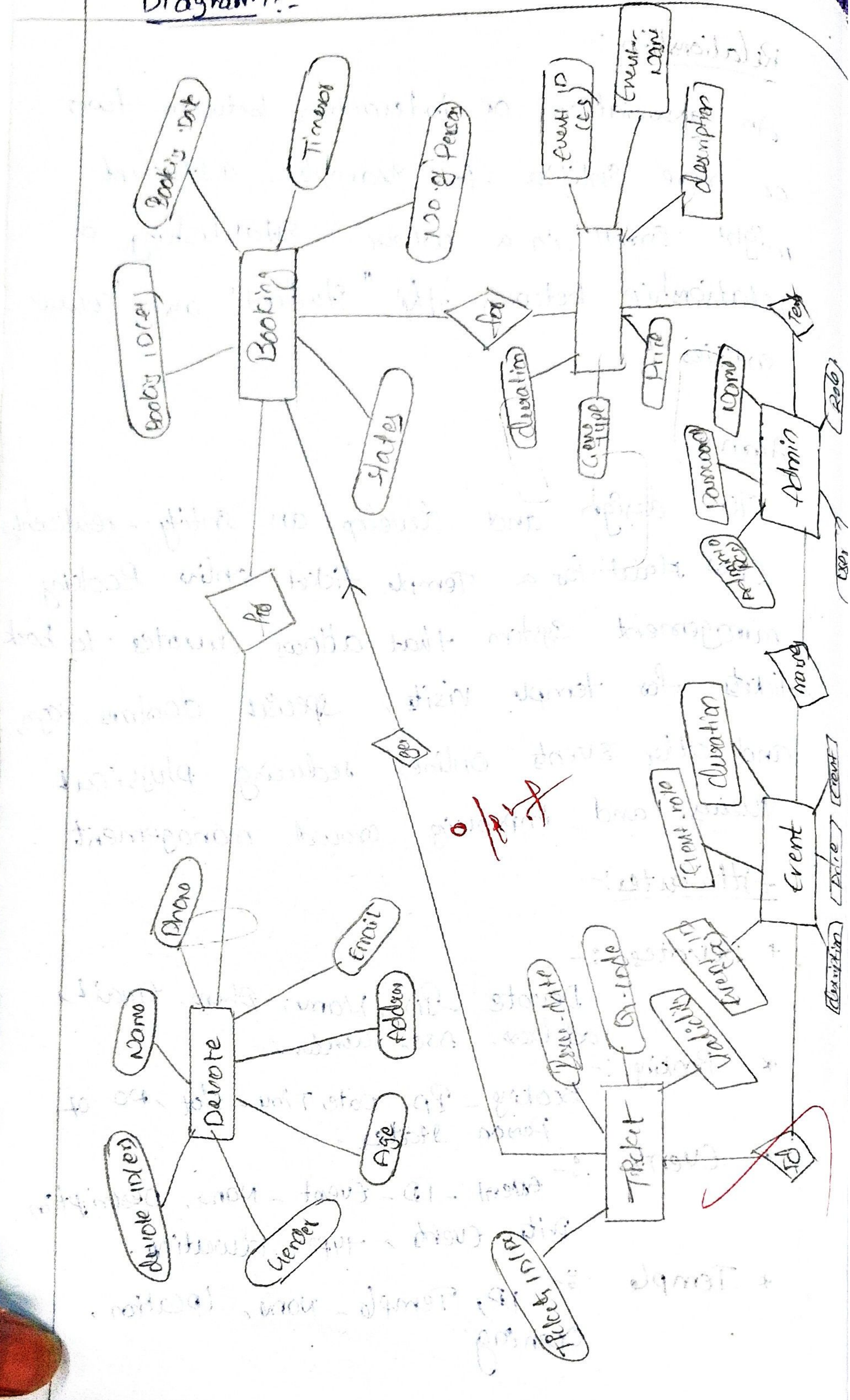
Aim :-

To design and develop an entity-relationship (ER) model for a Temple ticket online Booking management system that allows devotees to book tickets for temple visits, special darshans, pujas, and other events online, reducing physical queues and improving crowd management.

Attributes :-

* Devotees :-
  Devotee_ID, Name, Phone, Email, address, Age, Gender,
* Booking :-
  Booking_ID, Date, Time-slot, No. of Person States.
* Events :-
  Event_ID - Event_Name, Description, Pujo Event, Type, duration.
* Temple :-
  ID, Temple_Name, location, opening

Diagram:-


Booking
Devotee
Ticket
Event
Admin
for
gets
Status
Phone
Email
Address
Age
Gender
Name
Duration
Price
Password
Role
Description

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

Tickets :- 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 :-

* 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)	13+5=18
SIGN WITH DATE	

24/7/25

Result :-

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

21/7/25 Task 2

Task 2:
Generating design of other traditional database model

Aim:-
Creating hierarchical /network model of the device database by enhancing the sound abstract data by enhancing the sound abstract data by Performing following tasks using forms of inheritance.

2a. Identifying the specifying of each relationship find and form simples relations.

2b. check is a hierarchy / has-a-hierarchy and the perform generations and cor specialization relationship

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

2d. Remove the relations.

2e. perform SQL relations using DDL. DCl commands.

2a. Identifying the specifying of each relationship. find and form simples relations.

a. one-to-many (Devote → Booking):

* each devote can many booking.
* each booking is unique linked to one devote.

b. (Booking → ticket) one-one.
* one-booking generate one or more tickets
* Ticket independent Booking.

c. one-to-one [Booking ↔ event :
* each booking is made for one event.

d. One-to one or one to many Payment ↔ Booking:
each payment is tied to a booking

2. surplus relations :-
then we identify entity that might be redundant, merged or consolidated top normalization.
(a) Ticket may be merged with booking (1:1 or 1:n)
(b) event detailed redaunt in booking
(c) payment and booking 1:1 relation

(2b) check is a hierachy (has a - hierachy and performs generalization and for specilization relationship.
* Devote is a person (4 extended to priest, admin)
* event (sewa is a service of various temple services are modeled.
* Booking has a ticket.
* Booking has a relationship with uses and temple.

Generalization :
* person (general) → devote (specilised)
* service → event / sewa

Specilization :
Different types of users :
Devote user, Admin User

2c. Attribute Domain and check constraint :
* user phone number : VAR(VARR(15):

* check (phone_number) like [0-9]{10,15}]

* Booking . date_of_booking : DATE, CHECK.
(date_of_booking) >= current_DATE)

* Ticket . seat_count : INT, CHECK (seat_count
Between 1 and 20)

* event type : Enum ('Darshana', Pooja, abshickam)
check (type (IN()))

2d. Renaming Relations :

* User → Temple user
* Booking → Temple Ticket
* Ticket → Temple ticket
* Event → Temple event.

2e. SQL relations using DDL. DCL

Create TABLE Temple User (
User_id INT PRIMARY KEY,
Name VARCHAR(200) NOT NULL
phone_number VARCHAR(15) NOT NULL
check (phone_number ~ '^[0-9]{10,15}$'));

Create TABLE Temple (
temple_id INT PRIMARY KEY
Temple (temple_id),
event_type VARCHAR(20) check,
event_type IN ('DARSHAN', 'POOJA' Abhishekam))
event_date DATE
Constraint check_event_date check
(event_date >= CURRENT_DATE)
);

```
create TABLE Temple Booking (
    booking_id INT PRIMARY KEY,
    Temple user (user_id)
    event_id INT Reference
    Temple event (event_id),
    booking_date DATE Default
    Current_DATE.
);
```

o/p

```
CREATE TABLE Temple Ticket(
    ticket_id INT PRIMARY KEY,
    Booking_id INT References
Temple booking (booking_id,
seat_count INT (seat_count BETWEEN 1 and 10)
);
```

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	

Result: Thus the Hiearch model and network model has been succesfully created.

7/8/25

Task-3

Using clauses, operators and functions in queries.

Aim :-

To Perform query Procesing on a temple ticket online booking management system for different Retrieval results of queries using, DML DDL operations with aggregate functions, data function string function set clauses and operators.

Temple :-

Templeid	Name	location	Contact-No.
T1D01	Tirumala venkateswara	Tirupathi	9012476974
T1D02	meenakshi amma	madhurai	7989222725
T1D03	Kashi vishwanath	varanasi	7013982716
T1D04	Jagannadh temple	puri	9701224004
T1D05	golden temple	Amristsar	9908616499

visitor

visitor ID	f-name.	t-name	age	Email	Contact-no.
V001	Anil	Kumar	30	Anil@gmail.com	9044276954
V002	Alekh	Reddy	25	ABeth@gmail.com	9701224404
V003	Priya	Sharma	38	Priya@gmail.com	9908290010
V004	Aarav	patel	35	Aarav@gmail.com	7013982716
V005	Sneha	Rao	22	sneha@gmail.com	7989222725

Booking:-

Booking ID	Temple ID	visitorID	Booking-date	Ticket-Type	amount
B001	TID01	V001	2024-6-15	VIP	500
B002	TID02	V002	2024-6-16	General	100
B003	TID03	V003	24-6-17	General	100
B004	TID04	V004	24-6-18	VIP	700
B005	TID05	V005	24-6-19	general	200

Priest:

Priest ID	FName	Lname	Age	Email	Contact no.
P001	Ramesh	Iyer	50	Ramesh@gmail.com	9014276959
P002	Suresh	Sharma	45	Suresh@gmail.com	9701229404
P003	manish	pas	40	manish@gmail.com	7989221185

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

```
Select *
FROM Visitor
WHERE FName LIKE 'A%';
```

visitor ID	Fname	Lname	Age	Contact - NO.
V002	Arun	Reddy	25	9701224404
V004	Aaryan	Patel	35	7013982716

3- Add a column for special 'Special Seva' in Booking table.

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

Result

Table altered successfully

4. count the number of VIP ticket bookings :-

```
select count (*)
from Booking
where ticket_type = "VIP";
```

Result :-

count (*)
2

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	Tirupathi	9701224404
TIDO2	kashi	Varanasi	7013952711
TIDO3	Golden temple	Amritsar	7898222125

6. Select visitor ID and names of visitor who took and '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	Roo

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

(no result if all priest are assigned)

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

Result:-

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

14/8/25

Task : 4

Using functions in queries and writig Sub queries :-

Aim :-

To perform advanced query procesing and test it heuristic by designing optimal correlated and nested sub queries such as finding summary statistics 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_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.
```

Temple ID	Temple Name	location	Total Bookings
T1D01	Tirumala	Tirupathi	3
T1D02	meenakshi Amman kashi	madhurai	1
T1D03	jagannath temple	varanasi	1
T1D04		Puri	0
T1D05	golden temple	Amristar	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 Templet
JOIN Booking b
ON t.Templeted = b.temple ID
where b.Ticket_type = 'special'
Group by t.tName;
```

output:-

Temple Name	Total special Booking
Tirumala Venkateswara	1

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	Tirupathi	9701224404
TID03	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,
    v.age
    b.Booking ID,
    b.Booking-Date,
    b.Ticket-Type,
    b.amount
from visitor v
Join booking b
ON v.visitor ID = b.visitor ID
where v.age = 25;
```

Output:

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

4.5: To Retrieve the details of temples with no Bookings

```
Select *
from Temple
where Temple ID NOT IN
SELECT Temple ID
from booking
);
```

output :-

TempleID	Name	Location	Contact-NO
TIDO4	Jagannadh temple	Puri	9701224404
TIDO5	Golden temple	Amristar	9014276954

U6: To Retrieve the templeid, temple name, and visitor name for a particular visitorid given.

```
SELET  t.templeID,
       t.Name As templeName,
       v.FName As visitorName
From templeT t
   Join BookingT b
on t.TempleID = b.TempleID
   Join visitorT v
on b.visitorID = v.visitorID
where v.visitorID = 'V004';
```

output:

TempleID	TempleName	visitorName
TIDO1	Tirumala	Sneha

Result:

Thus the queries using function and nested subqueries executed for the temple booking management system.

21/8/25

Task NO :- 05

Aim :- Writing Join queries, Equivalent, and Recursive Queries

To perform advanced query processing and test heuristics using optimal correlated and Nested subqueries such as retrieving Summary statistics and ticket booking details for the online Temple Ticket Booking management system.

5.1) To Retrieve all temples and their available tickets.

```
SELECT tm.Name As Temple_Itr. Ticket ID.
tk.TYPE,
  From Temple tm
  JOIN Ticket tk ON tm.Temple ID = tk.Temple ID;
```

5.2) To list all Bookings along with temple name and devote name.

```
SELECT. b.Booking ID, d. Devote Name, tm. Name As
  Temple Name. tk. TYPE, b. Booking Date.

  From Booking b
  JOIN Devote d ON b. Devote ID = d. Devote
  ID.
JOIN Ticket TK ON b. Ticket ID = tk. Ticket ID.
JOIN Temple tm ON TK. Temple ID = tm. Temple
  ID;
```

5.1 output

Temple	Poja name	Price
Tirupathi	Subrabhatha Sella	300
Tirupathi	Archana	100
Kanchi	Abhishekam	500
madhurai	Speial darshan	250

5.2 output

Book-id	Devote	Poojaname	Book-date	time	Status
B101	Rajesh	Subrabhathasevа	22/07	6.Am	Confirm
B102	meena	Archana	22-07	9Am	Confirm
B103	Arjun	Abhisekam	23-07	7AM	confirm
B104	kavitha	Speial darshan	23-07	8.30Am	Confirm
B105	Suresh	Abhisekam	24-07	6AM	Confirm

5.3 output

Devote	Total Bookings
Rajesh	1
meena	1
Arjun	1
kavita	1
Suresh	1

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

```
SELECT tm.name AS "Temple Name",
count(b.Bookings) As Total Bookings
from Temple tm
(eft JOIN Ticket TK on tm.Temple ID = Tk.temple ID
LEFT JOIN Booking b on tk.Ticket ID = b.Ticket ID
Group By tm.Name;
```

5.4) To find all devotes who booked ticket for 'Tirupathi Balaji'.

```
Select d.Devotee ID.d.Devote Name, d.Contact no,
       tm.name As Temple
From .Devote d
JOIN Booking b on d.Devote ID = b.Devote ID
JOIN Ticket tk on b.Ticket ID = Ticket ID
JOIN Temple tm ON tk.Temple ID.tm.Temple ID
where tm.Name = 'Tirupathi Balaji';
```

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

```
SELECT.d.Devote ID.d.Devote Name.d.Devote ID
   Count(b.booking ID) As Total Tickets
From Devote d.
LEFT JOIN Booking b on d.Devote ID.b.Devote ID
Group By.d.Devote ID, d.Devotes Name.d.
Contact no;
```

5.4 output

Dev-Id	Fname	mobile no.
D105	Arjun	9845721999
D105	Suresh	7289654271

5.5 output

poojaname	total Bookings
Suprabhatha seva	1
Archana	1
Abhishekham	1
special darshan	

5.6 output

Temple	Cancel Bookings
Kanchi	1

5.6) To Retrieve the total number of 'special Darshan' ticket Booked temple-wise.

```
SELECT tm.name AS temple,
Count (b.BookingID) AS Special_Darshan_Count
FROM Temple tm
JOIN Ticket tk and tm.TempleID = tk.TempleID
JOIN Booking b ON tk.TicketID = b.TicketID
WHERE tk.TYPE = 'Special Darshan'
GROUP BY tm.name;
```

5.7) To Retrieve the temple details where tickets are still available (not fully Booked)

```
SELECT tm.TempleID, tm.name, tk.Type, tk.price
FROM Temple tm
JOIN Ticket tk ON tm.templeID = tk.TempleID
WHERE tk.AvailableSeats > 0;
```

5.8) To Retrieve devote and Booking details for devotes above 40 years.

```
SELECT d.DevoteID, d.Devote name, d.Age, b.Booking
ID, b.Booking Date, tm.Name AS Temple
FROM Devote d
JOIN Booking b ond. DevoteID = b.DevoteID
JOIN Ticket tk ON b.TicketID = tk.TicketID
JOIN Temple tm ON tk.TempleID = tm.TempleID
WHERE d.Age > 40;
```

S-1 output

Templ id	Name	location	Head Priest
1001	Tirupathi	Tirupathi	Ramanujan
1002	kanchi	kanchipuram	Natarajar
1003	madhurai	madhurai	Jashanathjee

S 8 output

Fname	Age	Book id	Date	time	Status
Suresh	55	B105	22-06	6Am	confirm

S-9 Output

Templ id	name	location	Head priest
T004	Rameshwaram	Rameshwaram	Vishnu devan

S-10 output

Temple	poojaname	Devote	Date	time
Tirupathi	Archana	meena	22.06	9Am.

5.9) To Retrieve temple details where no booking have been made

```
SELECT * from Temple
WHERE Temple ID NOT IN (SELECT tk. Temple ID
    FROM Ticket tk.
    JOIN Booking b ON tk. Ticket ID;
        b. Ticket ID);
```

5.10) To Retrieve temple ID, ticket ID, temple name and devotee name for a particular booking ID given

```
SELECT tm.temple ID, tk. Ticket ID, tm. Name As
    Temple name, d. Devote Name
from Temple tm
JOIN Ticket tk ON tm. Temple ID = tk. Temple ID
JOIN Booking b on tk. Ticket ID = b. Ticket ID
JOIN Devote d on b. Devote ID = d. Devote ID
WHERE b. booking ID = 'B002';
```

VEL TECH - CSE	
EX NO	1
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	18
SIGN WITH DATE	

21/8/25

Result:

Thus, the query using Join queries nested queries and equivalent queries were executed successfully for the online Temple Ticket Bms.

28/08/25

Task NO : 6

Aim:

To write and execute PL/SQL Procedures, functions, and Loops on Number theory (e.g. Prime Number, factorial) and Business Scenarios (eg: salary calculation, Discount calculation).

Algorithm :-

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) Busines Scenario example - salary Bonus calculation using function

1. Start.
2. create a function that accept emp - salary.
3. If salary < 2000 → 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 SERVEROUTPUT 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;
 /
```

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

(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 < 80000 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 ' || final_sal);
  END ;
  /
```

VEL TECH - CSE	
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	2
RECORD (5)	5
TOTAL (20)	17
SIGN WITH DATE	

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

4/09/25

Task: 7

Triggers, views and exception

Aim: To conduct Events, views, Exception on CRUD operations for restricting phenomenon in a temple online Booking management System.

a) To create a trigger in PL/SQL that automatic insert a new record in the booking auditable when a new Record is seated into the booking's table.

b) To create a view that displays the details of Bookings along with the user and temple details.

c) TO write a non- Recursive PL/SQL procedure to Retrieve even- number of Booking IDs registered for any temple.

a) Trigger

To create a trigger in PL/SQL that automatically insert a new record in the Booking audit table when a new record in inserted into the Bookings table.

```
CREATE or Replace Trigger
    insert - booking - audit
After insert on Bookings
    for ~~each~~ Row
        BEGIN
INSERT into Booking - audit (audit - id,
booking - id, user - id, temple - id, slot - ID,
new - Status, audit - ts)
Values
END;
```

b) View

To create a view that displays the details of Bookings along with the user and temple details.

```
CREATE or REPLACE VIEW BookingDetails AS
SELECT b.booking_id,
       u.user_id,
       u.full_name as user name,
       u.mobile As User mobile,
       t.temple_id,
       t.temple_name,
       s.slot_id,
       b.Status s,
       b.created_at
FROM booking b
JOIN users u on b.user_id = u.user_id JOIN
temples t on b.temple_id = t.temple_id.
JOIN slot s ON b.slot_id = s.slot_id;
```

To display:

```
SELECT * FROM BOOKINGDETAILS;
```

c) Procedure

To write a non-Recursive PL/SQL Procedure to Retrieve even member of Booking IDs Registered for any temple.

```
CREATE or REPLACE PROCEDURE
    Get even Booking IDs for Temple
      p_temple_id IN number,
      out_even_bookingids OUT
      sys.ODCINumberList
    ) AS
```

```
BEGIN
out_even_booking_ids := 
    sys.ODCINumberList();
FOR rec IN (SELECT booking_id
    FROM bookings
WHERE temple_id = in_temple_id
  AND MOD (booking_id,2) = 0)
loop
out_even_Booking_ids.EXTEND;
  out_even_bookings_ids(out_even_booking)
_ids.(COUNT) := rec.booking_id;
        END LOOP;
         END;
         /
```

Exception example:-

```
DEclare
temple_id Number (:= 101)
even_ids  sys.ODC.IN Number list;
 BEGIN
 Get even Booking IDs for Temple (temple.id,
 even_ids);
 for : IN ... Even ids  COUNT loop
DBMS_OUTPUT.Put_LINE ('Even Booking
    ID: ' || even_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)	20
SIGN WITH DATE	

4/9/25

Result: Thus the Triggers, views and Exceptions Experiment for Temple online Booking management System was successfully completed and result was verified.

11/09/25 Task no :- 8

CRUD operations in Document database.

Aim:- To perform mongose using MVC design on mongo DB for the online Temple Ticket BMS designing a document DB and performing CRUD operations like creating, inserting, querying - finding, updating, removing operations.

STEPS:-

Step 1 :- Install MongoDB.

2) Install Mongosh.

3) Add the MongoDB shell binaries location to your path.

open control panel -> system and security system.

* click advanced system settings.
* click environment variables.
* edit path and add the mongo DB shell binary location

4) confirm path by typing in command prompt
mongosh - help

5) open Mongo-DB server

6) Perform CRUD operations

CRUD operations for online Temple Ticket BMS

1) Create Collection

db.createCollection ("Temple Tickets")
{"ok" : 1}

output:

2) {
"acknadudged" : true,
"Inserted id":
object ("6516f112 60bbfe799300 f999")
}

3) {
": objecte id ("651Cf112 26ebbfe999")
" ticket id : 'TI001",
"Temple name": "Menakshi Amman Temple",
"location": "Madhurai",
"Booking date": "2025-09-10",
"Ticket price": 200,

4) {
"acknowledged" : true,
"Inserted Ids": [
objected ("651C917db.bbfe799301")
objected ("651ef112 00bb te 799302")
objected ("651ef126e bbfe799303")
]
}

5)

{

"Temple ticket id": "TPO1",

"Temple name": "Meenakshi Amman Temple",

"Location": "madhurai",

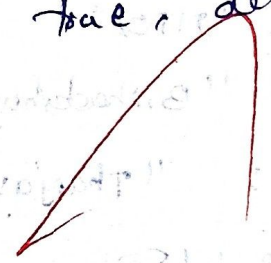
"Ticket price": 200

}

6) {"acknowledged": true, "matched count": 1,

"modified count": 1}

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



5) find All Document

db. Temple Tickets . find () Pretty ()

6) Update a Document

db . Temple Tickets . update. One ()

{ Ticket ID : "TIDOS" } ;

{ set : { Ticket price : 120, Devote name : "Ignesh Kumar" } }

)

7) Delete a Document

db . temple Tickets . delete one ({ Ticket ID : "TID02" })

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

11/9/25

Result :

Thus CRUD operations (Create, Read, Update, Delete) were successfully performed Using mongo DB with npm/ mongoose design.

18/09/25

Task: 9

CRUD operations on graph Database

Aim: To Design a graph database model for temple ticket booking using Neo4j and Graph and perform CRUD operations.

Algorithm :-

1. Define graph entities - Temple, devotee, ticket
2. Define relationships - devotee booked for ticket for - temple.
3. Design Graph QL schema type for these entities with relationships.
4. Use cypher queries for CRUD operations
   - * create Nodes
   - * Insert nodes
   - * Query graph
   - * Delete nodes.
5. Expose a Graph QL API using Neo4j-Graph Integration for client side CRUD operation Interaction.

Source code:

```
type temple {
    Temple id : ID,
    name : String ;
    location : string
    tickets : [ticket!] relationship
(type "for - Temple", direction : IN) }
```

output query

Ticket ID	Date	Seat Num	Status
tk1	2025-10-01	A10	Booked

find :

Ticket ID	date	Seat Number	Status
tk1	2025-10-01	A10	Booked

type ticket {
ticket ID = ID,
date string,
Seat number : string,
Status : string,
user : user ! @ relationship (type
"Booked : for", direction = "IN")
temple : Temple ! @ relationship (type :
"for temple", direction out }

Create Nodes :-

CREATE (t : temple {temple_id : '', Name : 'Somnath
temple', location : 'city A'});

CREATE (u : User {user ID : 'u1', name : 'Alice',
email : 'alice @ example.com'});

CREATE (tk : ticket {ticket ID : 'tk1', date
2025-10-01 : seat Number : 'A10' status :
'Booked'});

CREATE relationship :-

match (u : user {user id : 'u1'}), (tk : ticket)
({ticket id : 'tk1'}), (t : temple {temple ID :
'ts1'})

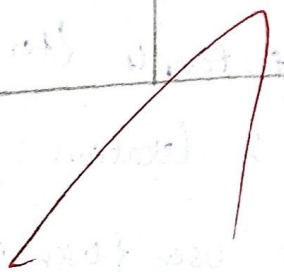
Create (u) - [: Booked - for] -> (tk);
Create (tk) - [: for - temple] -> (t);

Update:-

Ticket ID	date	Seat Number	Status
1	2025-10-01	A10	Cancelled

Delete:-

Ticket ID	date	Seat Number	Status
(none)			



Query ticket for temple:

match (t:temple {temple ID: '4'}) -[c:has temple].

[t.tk: ticket]

Return tk. ticket ID, tk. date, tk. seat number

tk. status.

find ticket for user :-

MATCH (u:user {user ID: 'ui'}) -(:Booked =for)

-> [tk. ticket]

Return (tk. ticket id. tk. date , tk. seat

number . t.k. status);

update ticket status:

MATCH (tk. ticket {ticket ID: 'TK 1'})

SET tk. status = 'Cancelled'

Return tk;

Delete ticket :-

MATCH (tk: ticket {ticket ID: 'TK1'})

DETACH DELETE tk;

VEL TECH - CSE	
EX NO	9
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	19
SIGN WITH DATE	

18/9/25

Result :

Thus the Program to design the graph Space and CRUD Operations is Verified and Successfully completed.

25/09/25 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.

Given Relation:

Temple Booking (
Booking ID, User ID, UName, Uemail, U contact, temple ID, TName, Tlocation, priest ID, Priest Name, Priest contact, TicketID, Booking date, darshan date, Timeslot, ticket type, Price Payment ID, Payment Mode, payment status. Transaction Date
).

Step (a): Apply functional dependencies → Normalize to INF.

1. User ID → UName, U email, U contact.
2. Temple ID → TName. ~~Priest~~ Temple contact
3. priestID → Priest name, priest contact
4. TicketID → Ticket type. Price.
5. Payment ID → Payment mode, payment Status, Transaction date.
6. Booking ID → User ID, Temple ID, priest ID, TicketID, Booking ID. darshandate, Time Slot Payment ID.

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

Step (b) :- Normalize using FD+ and α+.

Candidate keys:

* Booking ID (main unique identifier of booking).

from α+ (closure of candidate keys):

* Booking ID+ = (All attributes) → confirms Booking ID is a candidate key.

* User ID, Temple ID, Priest ID, Ticket ID, Payment ID also act as foreign keys with their own dependencies.

Step (c):- minimal cover / Canonical cover.

we Reduce FDs to minimal form:

1. User ID → U Name, U Email, U Contact.
2. Temple ID → T Name, T location
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.

* Already minimal and canonical → no redundant attributes.

Step(d) :-

Normalize to 2NF

Conditions for 2NF

* must be already be in 1NF
* No partial dependency.

Here, Booking ID, is the Primary key so no partial dependency exists.

The schema is in 2NF.

Step(e) :- Normalize to 3NF

Condition for 3NF

* must be already be in 2NF
* No transitive dependencies

Check:

* Booking ID → User ID → Uname.
* Booking ID → Temple ID → TName.
* Booking ID → Priest ID → Priest Name.
* Booking ID → Ticket ID → Ticket type, Price.
* Booking ID → Payment ID → Payment mode, Payment Status.

Final Simplified tables in 3NF

1. User table

User (User ID, [PK], Uname, Uemail, U contact)

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. [PK], Temple ID [PK], Priest ID [PK], Ticket ID. [FK], payment ID [PK], Booking date. darshan date, Time slot

Constraints:

* Primary key :

Booking ID, User ID, Temple ID, Priest ID, Ticket ID, payment ID

* Foreign keys:

→ Booking. User ID → User User ID

* Booking. Temple ID → Temple. Temple ID

* Booking. Priest ID → Priest. Priest ID

* Booking. Ticket ID → Ticket. ticket ID

* Booking. payment ID → payment. payment ID.

Unique Constraints

* price > 0
* payment status ∈ ('pending', completed, failed')
* Time slot within valid daychar timings

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

Result :- Thus the given relation for the Online Booking management system has bee normalized into simplified table up to 3NF.

9/10/25 Task-11

Design an online Ticket booking System Using oracle forums 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 form, define the data model that connect the database schema for the temple ticket booking system use.

* Temples
* Ticket types
* Booking details
* User Id Devotes
* payment information

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

Create menu :

menu provides the navigation structure for the booking system application

Steps to create menus in oracle form :

1. open oracle forms Builder
2. create a new menus from or use an existing one.
3. Add menu items for each major function:
   * Book tickets
   * 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 :

forms are used to capture, display and exit 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 file (for user Info), buttons (submit / cancel)

3. Use the Property P/Delete to Configure element Properties and link data blocks to database tables.

4. Write PL/SQL code for busines logic such as :
   * validating booking dates.
   * checking ticket availability
   * Processing Payments

Create Reports :

Report Provide Summarized information about booking and temple visit.

Steps to create :

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.

report :

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

4. Design a report layout with headers / footer / and detailer data columns.
5. Add Parameters to allow retrieving date, 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)	5
TOTAL (20)	18
SIGN WITH DATE	

9/10/25

Result: The system is successfully delivered to facilitate online 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

BOARD OF SECONDARY EDUCATION ANDHRA PRADESH

DADIPINENI BHARGAVI

PANAGANTI SAI SRINATH

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 (
 id INT PRIMARY KEY,
 name TEXT
);
```

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

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

CREATE TABLE payments (
  id SERIAL PRIMARY KEY,
  user_id INT REFERENCES users(id),
  coupon_id INT NULL REFERENCES coupons(id),
  amount NUMERIC,
  discount NUMERIC,
  final_amount NUMERIC,
  status VARCHAR(20),  -- 'success'|'failed'
  method VARCHAR(50),
  created_at TIMESTAMP DEFAULT now()
```

1.);
   Find coupon:

SELECT * FROM coupons WHERE code = 'WELCOME10';

2. Check total usage:

SELECT COALESCE(SUM(used_count),0) AS total_used

FROM coupon_usage WHERE coupon_id = 123;

3. Get user's usage:

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

4. Record successful payment:

INSERT INTO payments (user_id, coupon_id, amount, discount, final_amount, status, method)

VALUES (45, 123, 1000, 100, 900, 'success', 'card');

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

6. INSERT INTO coupon_usage (coupon_id, user_id, used_count)
7. VALUES (123,45,1)
8. ON CONFLICT (coupon_id, user_id) DO UPDATE
9. SET used_count = coupon_usage.used_count + 1;
10. **EXAMPLE: Concrete Coupon + Walkthrough**
11. Coupon row:

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

12. User places order:

user_id = 45

- cart_amount = 1200.00
- coupon_code = WELCOME10

Validation:

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

Assume tax 18% on final_amount (optional):

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

Payment simulation success.

SAMPLE PROGRAM OUTPUT(Console-style)

Applying coupon: WELCOME10

User: 45 Cart: 1200.00

Coupon valid — percentage 10% (max 150.00)

Discount applied: 120.00

Amount after discount: 1080.00

Tax (18%): 194.40

Final payable amount: 1274.40

Payment status: SUCCESS

Payment ID: PAY_20251017_001

Coupon usage updated for user 45 (used_count = 1)

user_id = 45

- coupon_code = WELCOME10

Validation:

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

Assume tax 18% on final_amount (optional):

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

Payment simulation success.

SAMPLE PROGRAM OUTPUT(Console-style)

Applying coupon: WELCOME10

User: 45 Cart: 1200.00

Coupon valid — percentage 10% (max 150.00)

Discount applied: 120.00

Amount after discount: 1080.00

Tax (18%): 194.40

Final payable amount: 1274.40

Payment status: SUCCESS

Payment ID: PAY_20251017_001

Coupon usage updated for user 45 (used_count = 1)

VEL TECH - CSE	
EX NO	04
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	18
SIGN WITH DATE	

Result:

16/10

Thus, the Building a backend Components that stores, offers & payment records & validate applies Coupons to order was executed.