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

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

ACADEMIC YEAR 2025-26 (SUMMER SEMESTER)

LAB RECORD NOTEBOOK

10211CA207 – DATABASE MANAGEMENT SYSTEMS

NAME: V. Tharun Krishna

VTU.NO: 30149

REG.NO: 24UECL0062

BRANCH: CSE (AI&ML)

YEAR/SEM: II/III

SLOT: S-10, L-12

INTERNAL EXAMINER EXTERNAL EXAMINER

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	: V. Tharun Krishna	**BRANCH**	: CSE (AI&ML)
VTU NO.	: 30149	**REG.NO.**	: 24UECL0062
YEAR/SEM	: II / III	**SLOT NO.**	: S-10, L-12

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

R.T. Thya 31/10/25

SIGNATURE OF LAB HANDLING FACULTY **SIGNATURE OF HOD**

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

INTERNAL EXAMINER **EXTERNAL EXAMINER**

INDEX

NAME: V. Tharun SUBJECT: DBMS - LAB

STD: DIV: ROLL NO: 30140 SCHOOL:

INDEX

No	Date	Title	Page No.	Marks	Faculty Signature
1.	24/07/25	Conceptual Design after Formal Technical Review	1-3	18	24/7
2.	31/07/25	Generating design of other traditional database models	4-9	18	31/7
3.	07/08/25	Developing queries with DML Single-row functions and operations	10-12	18	7/8
4.	14/08/25	Developing queries with DML Multi-row functions and operations	13-16	18	14/8
5.	21/08/25	Writing Join Queries, equivalent, and/or recursive queries	17-20	18	21/8
6.	28/08/25	Writing PL/SQL using Procedures, Function	21-24	19	28/8
7.	4/09/25	Writing PL/SQL using Loops	25-28	20	4/9
	11/09/25	Normalizing databases using functional dependencies up to BCNF	29-31	19	11/9
	18/09/25	Backing up and recovery in databases	32-33	18	18/9
10.	25/09/25	CRUD operations in Document databases	34-37	14	25/9
11.	09/10/25	CRUD operations in Graph databases	39-41	18	9/10
12.	16/10/25	Micro Project: indexing various devices in the IoT platform	42-47	18	16/10

completed

201

Total Marks: 219 / 240

Signature of Faculty

Date: 24-7-25

Task-1

A Temple ticket online booking management system

A Temple ticket online booking management system allocat devotes To book tickets for temple visits, special darshan, poojas, and others events online, reducing physical queues and improving crowd management. There systems often include features like date and time slot selection, payment procelling and the generation of Tickets or passes.

Entity:-

The real-word object or concept that can be definitely identified. examples included students, courses or products

Entity set:-

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

Attributes:-

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

Relationships:-

An Association or interaction between two or more entities for Example, a student might enroll in a course establishing a relationships between the "Student" and "course" entities

AIms

To design and develop an Entity - relationship (ER) model for a Temple Ticket online Booking mangement system That allows devotes to book Tickets for Temple visits, special darshans, Poojas, and others eveents online, reducing Physical ques and improving crowd management.

Attributes:-

- Devotee :- Devotee- DP, Name, Phone, Email, adress, AGE, Gender
- Booking :- Booking - DD, Date, Time-Slot, No. of Person, Status
- Events :- Event-ID, Event- Name, Description, Price, Event-type, duration.
- Temple :- Temple - ID, Temple- Name, location, Opening hours
- Payment :- Payment - ID, Amount, method, States, Dates
- Ticket :- Ticket- ID, issue- Date, QR-code, vasicility
- admin :- Admin - ID, Name, username, Password, Rac

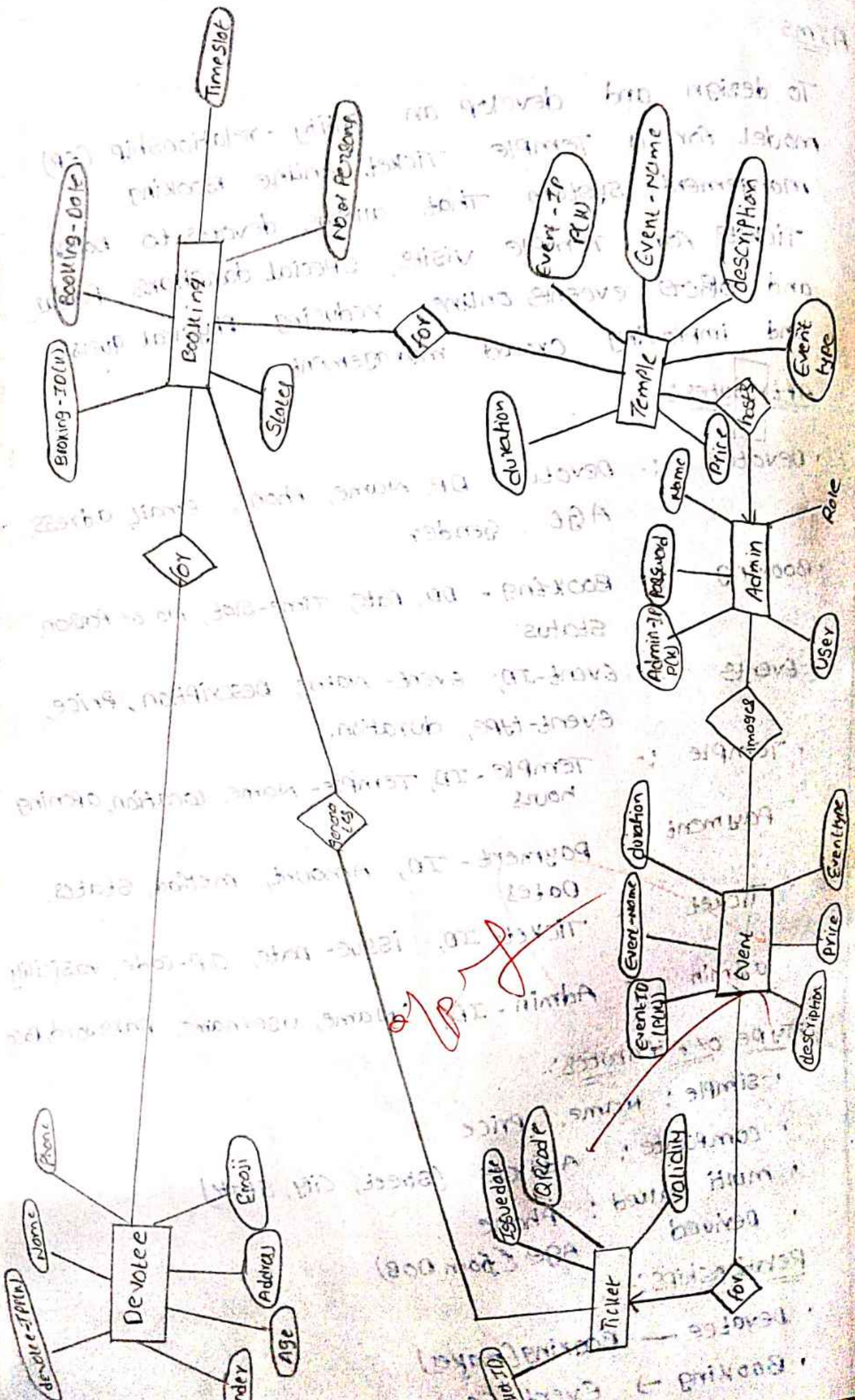
Type of Attributes:-

- simple : Name, Price
- composite : Address (Street, City, State)
- multi valued : Phone
- Derived : Age (from DOB)

Relationships:-

- Devotee - Booking (makes)

Diagram


Booking
Booking-ID(K)
Booking-Date
TimeSlot
No of Persons
Slots
for
Temple
Event-ID P(K)
Event-Name
description
Event type
duration
Price
hosts
Admin
Admin-ID P(K)
Password
Name
Role
User
images
Event
EventID (PK)
Event-Name
duration
Event type
Price
description
for
Ticket
Issue date
QR Code
validity
Devotee
Devotee-ID(K)
Name
Phone
Email
Address
Age
Gender
for
Bonga (Ds)

- Booking → Payment (has)
- Booking → (generates) Ticket
- 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)	13+5=18
SIGN WITH DATE	

24/7/25

Result:- The ER diagram for the temple Ticket Booking system was sucessfully designed, showing all entities. attributes, and relationship with correct cardinalities for database implementation.

Date:- 31/7/25 Task-2 4

Task 2:- Generating design of other traditional database model:

Aim:-

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

2a. Identify the specify of each relationship And and form surplus relations.

* Relationship :- Temple conducts Pooja (one to many)

* Specificity : one temple can conduct many Poojas. but each Pooja is associated with only one temple.

* Surplus relation: No surplus relation is needed for this relationship since it is already one to many.

Relationship :- Devotee books ticket (many to one)

* 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 relation ship since it is already many-to-one

Relationship :- Ticket belongs to Pooja (many to one)

* Specificity : A ticket is associated with one Pooja but a pooja can have many tickets

* Surplus relation :- No surplus relation is needed

Relationship :- Booking has payment (one-to-one)

* Specificity: one Booking is associated with one payment, and each payment is for one Booking

* Surplus relation : Not required

Relationship: Admin manages temple (one-to-many)

* Specificity: one admin can manage multiple temples but each temple is managed by one admin
* Surplus relation: No surplus relation is needed

Based on the specificity analysis, all relationships are correctly represented. No additional surplus relations are necessary.

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

Generalization:

In the online temple ticket booking management system, generalization can be applied to the entities devote and admin as both store common attributes.

Entities :-

* Devotee
* Admin

common attributes:

- Name
- Email
- Contact-No

* Generalised Entity

- Person
- PersonID
- Name
- Email
- Contact-No

subclasses :

Devote :- Inherits from person and adds attributes like devote, gender, Age, Address

Admin: Inherits from Person and adds attributes like Admin ID, password

By introducing a generalization, we reduce redundancy, enhance maintainability, and centralize shared data

Specialization:

In the same system, specialization can be applied to Pooja, where different types of Pooja exist with specific features

Entity

• Pooja

Specilized subtypes:

' Archana Pooja: additional attributes like Flowers mantras, Duration,

' Abhishekam Pooja' attributes like water, milk etc. This specialization Provides flexibility to represent different Pooja types with their own specific attributes while still maintaining general Pooja details in the parent entity,

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

let's take the age attribute in the Devotee entity ~~ats~~ as an example

Domain of age:

The valid age for booking should be between 10 and 100 years

Constraint applied:

A check constraint is applied to ensure age is entered between 10 and 100

Action taken:

A constraint is applied to ensure the value of age is not below 10 or above 100

• if invalid age is entered, The system will reject the record.

2d Rename the relations:

remaining relations and coloumns can help improve clarity and standardize naming

Eample 1:

Old coloumn name, Contact-No in devotee
New coloumn name: Phone-no

Action taken,

coloumn renamed sucessfully

updated table structure:

Name	Null	Type
DevoteeID	Not Null	varchar 2(10)
F Name		varchar 2(20)
L Name		varchar 2(30)
Age		Number (3)
Gender		varchar2(10)
Address		varchar2(50)
Email		varchar 2 (40)
Phone-No		Number

Example 2:

Old table name: Ticket

New table name :- Ticket details

Action taken:

Table renamed successfully

2e Perform SQL Relation using DDL. DCL commands DCL (Data control language) is used to manage Privileges on database objects

Step1: A new user named meena was created to access the system

Step2:- Privilages were granted to meena to create sessions and resources,

Step3: meena logged in and created a table for storing temple feedback

Step4:- The system neer granted privilages on the temple table to meena

Comands Performed:

- User Created
- Resource role granted
- Create session granted
- Table Created

- All Privilages granted on temple table

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	

31/07

Result:- Thus the Hierarchical model and network model for online temple Ticket Booking management System has been successfully Created.

Date :- 7/8/25

Task-3 Using clauses operator and function in Queries

Aim :- To perform the querry processing on database for different retrival results for of queries using DML, DRL operations with aggregate, data, string in built functions, set clauses, and operators for the online temple ticket booking management system.

Table structures and sample data

Temple:

Temple ID :-	Name	location	Deity
T001	Tirumala	Tirupati	lord venkateswara
T002	meenakshi	madhurai	Goddess mahalakshmi
T003	kasi	varanasi	lord Shiva

Devotee:

Devotee id	Fname	Lname	Age	Gender	contact NO.
D001	Arjun	Reddy	30	m	9842732810
D002	Akhila	devi	28	F	9765237821
D003	Ramesh	naidu	35	m	9386714859

Donation

Donation ID	DID	TID	Donation date	Amount
DON001	D001	T001	1-8-25	5000
DON002	D002	T002	2-8-25	2000
DON003	D003	T003	5-8-25	3000

Priest

PriestID	TempleID	FName	LName	Specialization
P001	T001	Raghav	Sharma	Arachna
P002	T002	krishnan	Iyer	Abhishekam
P003	T003	Suresh	Babu	Flomon

Queries and results:

3.1 To retrive the location of a particular temple by its TempleID

SQL

Selection location From Temple where Temple ID = "T008"

Result: location - varanasi

3.2 To retrive the devotees whose name start with A

SQL:

SELECT " From Devotee WHERE FName Like 'A%'.

Result:

DevoteID	FName	LName	Age	Gender	Contact no
D001	Arjun	reddy	30	M	9542231210
D002	Akhila	devi	28	F	9760231782

3.3 To count the number of special tickets booked

SQL:

SELECT COUNT(") From Booking WHERE Ticket type 'Special':

Result:

Count = 2

3.5 To display the temple details for Temple ID's T001, and T003

SQL

SELECT * FROM TEMPLE WHERE TEMPLEID IN ('T001'

Result:

Temple ID	Name	Location	Deity
T001	Tirumala	Tirupati	lord venkateswara
T003	Kasi	varanasi	lord Shiva

3.6 To find the devotees who have not made any donation

SQL:

SELECT * DEVOTEEID FROM DEVOTEE WHERE DEVOTEE ID NOT IN (SELECT DEVOTEE ID' FROM DONATION):

Result: Devote ID

VEL TECH - CSE

EX NO	
PERFORMANCE (5)	3
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	3
TOTAL (20)	15
SIGN WITH DATE	13 + 5 = 18

Result:- Thus the query processing on the database for different retrived results using clauses operations and functions in the online temple ticket booking management system has performed sucessfully

Using functions in querries & writting sub queries:

Aim:- To Perform advance query processing & test its hevesities by designing optimal corrected and nested subquires such as Anding Booking management system

4.1 : To retrive all temple; details including the count of bookings for each Temple:

```
SELECT : TEMPLE.ID
         NAME AS TEMPLE NAME
         LOCATION
         CONTACT-NO.
  COUNT (b. BOOKING ID) AS TOTAL BOOKINGS FROM
TEMPLES.
 LEFT JOIN BOOKING ID
 ON t. TEMPLE ID = b TEMPLE ID
 GROUP BY t. TEMPLE ID, t. NAME, t. LOCATION, t CONTACT NO
```

4.2 To retrieven the total no of special ~~of~~ Booking in a temple-wise manner

```
SELECT E. NAME AS TEMPLE NAME,
COUNT [*] AS TOTAL SPECIAL BOOKINGS
FROM TEMPLE t
JOIN BOOKINGS
ON t. TEMPLE ID = b. TEMPLE ID
WHERE b. TICKET - TYPE = 'SPECIAL'
GROUP BY t. NAME;
```

4.3 To Reviver the details of temples when booking include 'VIP' tickets

```
select *
from Temple.
WHERE Temple ID INC
SELECT Temple ID
from Booking
WHERE Ticket-Typ = 'VIP'
y;
```

OUTPUT:

TempleID	Name	location	contact NO:
T1 D01	Trimula	Tirupati	9701224404
T1 D03	Kashi	varanasi	7013982716

o/p

4.4 To retriever visitors and booking details of visitor who care above 25 Year 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
    Total Booking
ON. v. visitor ID = b visitor ID
WHERE v. Age =25;
```

OUTPUT:

Visitor ID	visitor name	Age	Booking ID	Amount
V001	Anil	30	13001	100
V003	Prigam	28	13003	100
V004	Aarav	35	13004	900

4.5 To retrieves the details of temples with no booking

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

OUTPUT:

Temple ID	name	location	contact no:
TID04	Tagonnata	Puri	9702244044
TID05	Golden	Amritsar	9014276954

4.6 To retrieve the temple id, temple name and vistor name for a particular visitor id given

```
SELECT t. Temple ID,
       t. name AS Temple name
       U. FNAME AS vistor name
FROM temple t.
JOIN Booking b
ON t. TEMPLE ID = b. temple ID
```

Output:

Temple ID	Temple Name	visitor name
T1001	Tirumala	S neha

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, joins and nested subqueries were sucessfully excuted for the temple ticket online booking management system

writing join queries, equivalent and or recursive queries:

Aim:- To Perform advanced Query processing and test its heuristic using join queries equivalent Queries and recursive Quires for the temple ticket online Booking managment system

Queries & outputs:-

5.1 To restrieve all temples & Their Pooja

Seleted t.name its temple p Pooja name from temple t

Join Pooja p on t.temple ID = p.temple ID;

5.2:- To list all booking along the devotee and Pooja details

Select b booking ID, d if name AS, devotee,

D. Pooja name; b-booking date, bislot time b.states.

from Booking b

Join Devotee d. ON b devotee ID= d devotee JOIN Pooja

P-ON b.pooja ID = P Pooja ID:

5.3:- count the number of bookings made by each devotee

Select d.F name AS devotee, count AS total bookings

from Devote d.

left Join booking b ON d devotee: d=b devoted Group By

d.F name e;

5.4:- To And all devotees who booked 'Abhishekam'

SELECT d.devotee id, d.f name Id= b devoted from devoted

From devoted

Join Booking on a d.devotee Id. b. devotee id

Join Pooja P. ON b pooja ID = P Pooja id WHERE 'P' Pooja name = Abhishekam

5.5 To retrive all Poojas and the no of times they were booked

SELECT P.Pooja Name, COUNT (b.booking ID) AS total Bookings

From Pooja P

left Booking b. ON P. Pooja ID = b.Pooja ID GROUP BY P. Pooja name,

5.6:- To retrive the total no 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 = b Temple ID

~~Joing~~

WHERE b status = cancelled

GROUP BY t.name;

5.7 To retrive temple details where booking were successfully

SELECT Distinct t. temple name, t Name t. location, t. head, first

from temple t.

S.5 OUTPUT

Pooja Name	Total Bookings
Suprabhata seva	1
Arachana	1
Abhishekam	2
Special darsahan	1

S.6 OUTPUT

Temple	Canacelled Booking
kanchi temple	1

S.7 OUTPUT

Temple ID	Name	location	Head Priest
T001	Tirupatitemple	Tirupati	Ramunjya
T002	kanchi	Kanchipuram	Narayan-DAS
T003	madurai	madurai	Subramanyam

S.8 OUTPUT

F Name	Age	Booking ID	Booking Date	Stime slot
Suresh	55	B105	24-6-23	6:30A Confirm

S.9 output

Temple_ID	name	location	Head Priest
T004	Rameshwaram Temple	Rameshwaram	Vishnu Devam

S.10 output

Temple	Pooja name	Devotee	Booking Date	Slot time
Tirupati	Arachna	meena	22-7-23	9:00

Result:-

System

Queries

S.9 output

Temple_ID	Name	location	Head Priest
T004	Rameshwaram Temple	Rameshwaram	Vishnu Devam

S.10 output

Temple	Pooja name	Devotee	Booking Date	Slot time
Tirupati	Arachna	meena	22-7-23	9:00

VEL TECH - CSE

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

27/8

Result:- This the temple ticket online booking managment system queries using Join queries, equivalent & recursive queries was successfully excuted.

Date: 28/8/25 Task-6

21

Procedures, Function AND Loops

Aim:- To write programming using PL/SQL procedures, functions & loops on business scenarios like online Temple ticket booking management system

1. write a PL\SQL block that calculates the average age of devotees and displays the result.

SQL:

```
Declare
total -age Number:=0;
num - devotees Number:= 0;
avg- age Number :=0;
BEGIN
FOR d_rec IN (SELECT Age FROM Devtee) LOOP
total -age := total -age+ d - rec - Age;
num - devotees := num - devotees +1;
ENDLOOP
If num- devotees >0 THEN
avg -age := total -age/num -devotees;
END IF;
DBMS- output. Put- lane ('Total Devotees:'
  ||num - devotees)
DBMS -output. Put - lane ('Total Age:' ||Num total-age)
DBMS -output. Put - lane ('Average age:'|| avg -age);
END;
```

Output:-

Total Devotees : 14

Total age : 560

Average age : 40

Sample input & output:

Booking ID : B102

Booking ID : B205

Booking ID : T01

Booking Date : 2025 -08-28

Visit Date : 2025 -09-10

Ticket Type : VID

Amount : 500

Output:-

Booking record inserted Sucessfully

2. write. a PL\SQL block Inserts a new devotee booking record into the Ticket_Booking Table

SQL:

```
DECLARE
V-Booking ID VARCHAR2(6):='&Booking ID';
V-Devotee ID VARCHAR 2(6):='&Booking DevoteeID';
V-Devotee ID VARCHAR 2(6):='&TempleID';
V-Booking Date DATE := TO-Date
('&Booking date', 'YYYY-MM-DD');
V-Ticket Type VARCHAR 2(20):='&Ticket Type';
V-Amount Number :=&Amount;
BEGIN
INSert into Ticket-Booking
(Booking ID, Devotee ID, Temple ID, Booking Date, visit Date
Ticket Type, Amount)
values (V-Booking ID, V-Devotee ID, V-Temple ID, V-Boo
king, Date, V-visit Date
COMMIT;
DBMS-OUTPUT. PUT-LINE ("Booking record Inserted
successfully");
Exception
  WHEN OTHERS THEN
DBMS OUTPUT. PUT-LINE
("ERROR:" || SQL ERROR);
ROLL BACK;
```

OUTPUT:

No. of bookings in temple 101 : 12

END;

3. Create a function that returns the total number of bookings for a particular temple.

SQL:

```
CREATE OR REPLACE FUNCTION
 Get total Bookings in Temple
  (P_ Temple ID VARCHAR)
  RETURN NUMBER IS
  V_ Total Bookings Number := 0;
 BEGIN
   SELECT count (*) IN To v_total Bookings
 FROM Ticket_Booking
 WHERE Temple ID = P_temple ID;
 Return V_ Total Bookings;
 Exception
   WHEN NO_DATA_FOUND THEN
    Return 0;
 WHEN others THEN
       Return -1;
 END Get Total Bookings an
 Temple;
```

Usage Example:

SQL:

```
Declare
    res Number;
BEGIN
  res = Get Total Bookings in temple ('T01');
```

OUTPUT:

Yaml

Even- Numbered Booking ID: 102
Even - Numbered Booking ID: 104
Even - Numbered Booking ID: 106

END;

4. write a non recursive PL/SQL Procedure to retrive even numbered Booking ID's

SQL:

```
CREATE OR REPLACE PROCEDURE
Get Even Booking IDS IS
BEGIN
  for b- rec IN (Select Booking ID from ticket-Booking
WHERE MOD (TO-Number (Booking ID), 2) =0) loop
DBMS - OUTPUT. PUT-LINE ('Even Numbered Booking ID:' ||
b-rec.BookingID);
  END loop;
END Get even Booking IDS;
```

Execution Example:-

```
SQL
  BEGIN
    GET Even Booking IDS;
  END;
```

VEL TECH - CSE	
EX NO	6
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	4+5 = 19
SIGN WITH DATE	

Result:-

This the PL/SQL procedures, functions and loops on online temple ticket booking managment system were sucessfully written excuted, and verified

Date: 4-9-25 TASK:- 7 25

Triggers, views and exceptions

Aim :-

To conduct events, views, and exception handling on CRUD operations for the simple ticket online booking management system.

a) Trigger

Requirment:

When a new booking is inserted into the booking table, automatically insert a corresponding record into the ticket table with default status as 'Active'.

SQL

```
CREATE OR REPLACE Trigger
trg_insert_ticket
After Insert ON Booking
FOR EACH ROW
BEGIN
    Insert Into Ticket (TicketID, BookingID, Devotee
    New ID, Age, Gender, QR codes, status)
    VALUES (
Seq_ticket_nextval.
sequence for ticket IDS
    : New Booking ID,
    'Unknown',
Place holder
    NULL,
- given at booking time
```

1. OUTPUT :-

Trigger Created

2. OUTPUT :-

~~Procedu~~

View Created.

3. OUTPUT :-

Procedures Created

Bookingid	Confirmation NO	visitor Name	Temple name	Booking Da
401	BK205105	John	Brihadeswar Temple	5-Dec-25
355	BK206106	Meena	Meenakshi Temple	20-Nov-25
500	BK207107	Priya Singh	Golden Temple	1-Oct-25
300	BK208108	Rajesh	Sumanth Temple	15-Sep-25

```
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 DetailsView
AS
SELECT
  b.BookingID
  b.BookingRef,
  u.Name AS Devotee type,
  t.Name AS Temple type,
  dt.Name AS DarshanType
  s.Starts TS AS Slotstart
  s.END TS AS Slot start,
  s.END TS AS Slot END,
  b.qty,
  b.Amount
  b.Status
FROM Booking b
JOIN users u on b.user ID = u.user ID
JOIN Temple t on b.Temple ID = t.Temple ID
```

4 output:

1 row inserted

1 row inserted

5. output:

Total Bookings made Today : 1

1 row updated

PL/SQL trigger completed successfully

6. output:

Booking ID	Name	Booking Date	Time Slot	No. of Persons
1305	Sai	3-SEP-25	morning	1

7. output

Booking 1305 canceled successfully

8. output

Error : ORA - 20001 : Booking ID Not found

4 output:

1 row inserted

1 row inserted

5. output:

Total Bookings made Today ; 1

1. row updated

PL/SQL trigger completed successfully

6. output:

Booking ID	Name	Booking Date	Time Slot	No. of Persons	Tot
1305	Sai	3-SEP-25	morning	1	30

7. output

Booking 1305 canceled successfully

8. output

Error : ORA - 20001 : Booking ID Not found

SQL

... To query

SELECT * FROM Booking Details view;

c) Non- Recursive PL/SQL Procedure

Requirement:

Rebrieve even- numbered Booking IDS for any given temple
(similar to even player IDS)

SQL

```
CREATE OR RERLACE procedure
get Even Booking IDSFO
  In - temple - id Num
  Out - even - booking - ids OUT
SYS. OD (INumber List
) AS
Begin
  out - even - booking - ids := 
  SYS. OD( INumber List( );
  For  rec IN C
      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;
ENDloop;
END;
```

Execution Block:-

```
sql
DECLARE
  temple-id.Number := 101;
  even-booking-ids SYS.OPCINUMBERLIST.
BEGIN
  get Even Booking IDS for Temple (temple-id, even-
  booking-ids);
FOR i IN 1... even-booking-ids COUNT LOOP DBms-outp
.PUT-LINE ('Even Booking ID:' || even-booking-ids(i)
END loop;
END;
```

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

4/9

Result:- Thus, the Triggers, views and exceptions for the temple Ticket online Booking mangament system were sucessfully implemented and verified.

OUTPUTS:

① Insert Temples

{
"acknowledged" : true
"Inserted Id" : objectId ("65 1d1 c96ebbfe7993a
}

{ "acknowledged", true,
"Inserted Ids" : {
object Id ("651d1CC36ebbfe7993adf911"),
object Id ("651d1 cc36 ebbfe 79939df912"),
}
}

② Insert Ticket Booking

{
"acknowledged" : true,
"InsertedId" : object Id ("651d1c606ebbfe7993adf913")

③ Query All Temples

{ "_id" : object Id ("651d1 ca66 ebbfe 7993adf910"),
"Temple ID" : "TTD01",
"Name" : "meenakshi Amman Temple",
"location" : "madurai",
"contact" : 9876543210
"Timings" : "6:00Am - 9:00pm
}
{ "_id : Object Id (651d1 ca66ebbfe7993adf911),
"Temple ID" : "TTD02"

Task-8

Date: 11-10-25 29

CRUD operation in document database

Aim:- To design a document database for online Temple Ticket Booking management system using mongo db and perform CURD operation: Create, Read, Update, Delete tickets, temples, and user bookings

Steps:

* Step 1: install mongoDB community Edition
* Step 2:- install mongo shell
* Step 3:- configure PATH environment variable to include mongo bin binary
* Step 4:- confirm installation
* Step 5:- start mongo DB server
* Step 6:- perform CRUD operation on Temple Ticket Booking Database

Example Document Structure:

{ → collection: Temples

"Temple ID": "TT001",

"name": "Meenakshi Amman Temple",

"location": "madurai",

"contact": 9876543210

"Timings": "6:00 AM - 9:00 PM",

Collection: Tickets

{

"Ticket ID": "TKI001",

"Temple ID": "TT001",

"User name": "John Doe"

"name": Ramanathaswamy Temple,

"location": Rameswaram,

"contact": 9765432109

"Timings": "5:00Am-8:00Pm"

}

{

"_id": ObjectId("651d1cc36ebbfe7993edf912"),

"TempleID": "TID05",

"name": Brihadeeswarar temple,

"location": Thanjavur,

"contact": 9654321098,

"Timings": "7:00Am-8:30pm"

}

④ Query Ticket Booking by Username

"_id": ObjectId("651d1cc36ebbfe7993edf913"),

"TicketID": "TKID01",

"TempleID": "TID01",

"UserName": "JohnDoe",

"BookingDate": "2025-09-10",

"NumberofPerson": 3

⑤ Update Temple contact

{

"acknowledged": true,

"matchedcount": 1,

"modified count": 1

}

"Booking Date": "2025-09-10",

"Number of Persons":

CRUD Operations

1. Create collection

db.Create collection ("Temples");
db.Create collection("Tickets");

2. Insert Documents

insert Temples

db-Temples. insertone ({
TempleID : "TT001",
Name : "meenakshi Ammon Temple",
location ; "madurai"
contact : 9876543210
Timings : "6:00Am -9:00Pm".
3});

db. Temples. insert many ([
{ TempleID : "TT002", Name: "Ramanathaswamy Temple", location: "Rameswaram, contact: 9765432109, Timings : "5:00Am-8:00Pm"},
{TempleID: "TT003", Name: "Brihadeeswarar Temple", location: "Thanjavur", contact : 9854321098, Timings: "7:00Am-8:30Am"}
]);

Insert Ticket Booking

db. Tickets. insert one ({
Ticket ID : "TKT001",
Temple ID : "TT001",
Username : "John Doe",

6) Delete a Ticket Booking

{
"acknowledged" : true,
"deleted count" : 1

⑦ Delete a Temple

{
"acknowledged" : true
"deleted count" : 1

Booking Date : "2025-09-10",

Number of Persons : 3

3});

3) Read /Query Documents

Find All Temples

db.Temples.Find().Pretty();

Find ticket by User

db.Ticket.Find({username: "JohnDoe"}).Pretty();

4) Update Documents

* Update Temple contact Number

db.Temples.updateone(
{TempleID: "TID01"},
{ Set : { contact : 9998887776 }
});

5) Delete Documents

* Remove Ticket Booking

db.Tickets.deleteone({ Ticket ID : " TKID01"})

* Remove a Temple

db.Temples.delete one({TempleID : "TID03"});

Result:

This, using mongoDB with mongoose we successfully designed a document database for an online Temple Ticket Booking managment system and performed CRUD operations.

* Create Temple & Tickets
* Read /Query temple and Ticket records
* Update temple details
* Delete ticket booking and temple records

VEL TECH - CSE

EX NO	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	11.9.25

CRUD Operations in Graph Databases

Aim:- To Perform CRUD operations Live creating, inserting Querying, updating, and deleting on graph spaces for an online Temple Ticket Booking management system.

Steps:-

1. open https://neo4j.com/cloud/platform/aura-graph-databases/?ref=docs-get-started-drop down
2. Click start free → continue with google → open
3. Copy the downloaded password file info and login to the Auto console.
4. Start executing queries in the Neo4j Browser

1. Create nodes with properties:-

Each node represents an entity like Devotee, Temple, and 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 {TempleID: 'T001', Name: 'Srivenkatesa Temple', location: 'Tirupati', capacity: 5000}) Return t

* Create a Booking node

CREATE (b: Booking {BookingID: 'B001', DevoteeID: 'D001', TempleID: 'T001', Date: '2025-09-10', No of Tickets: 2}) return b

output:

1. Creating Nodes

* Create Devotee Node:

d. Devote ID	d. Name	d. Age	d. Phone	d. email
D001	sai	30	9876543210	sai@gm

* Create Temple Node :-

t. temple ID	t. Name	t. location	t. capacity
T001	Sri venkateswara Temple	Tirupati	5000

* Create Booking Node:-

b. Booking ID	b. Devotee ID	b. Temple ID	b. Date	b. Node
B001	D001	T001	25-09-10	

2. Create Relationships

* Devotee MADE Booking

* Shows nodes and relationship [MADE] connecting Devotee to Booking

* Booking FOR Temple:-

* Shows nodes and relationship [FOR] connecting Booking to Temple.

3. Display All Nodes

Node Type	Properties
Devotee	{DevoteeID: 'D001', Name: 'So Age: 30, Phone: 9876543210, 'sai@gmail.com'}
Temple	{ TempleID: 'T001', Name: 'Sri Ve Temple, Location: 'Tirupati', Capa
Booking	{ BookingID: 'B001', DevoteeID: ' TemplesID: 'T001', Date: '2025- No of Tickets: 2}

4. Retrieve particular Booking

b.BookingID	b.DevoteeID	b.TemplesID	b.Date	b.No of
B001	D001	T001	2025-9-10	2

5. Update particular Devotee

d.DevoteeID	d.Name	d.Age	d.Phone	d.Email
D001	Sai	31	9123456780	sai@gmail

6. Delete particular Devotee

* No Result returned 'but the node is deleted successfully
* Query MATCH (b: Booking {BookingID: 'B001'}) Return b returns no rows

2. Create Relationships

* Devotee makes Booking

MATCH (d:Devotee {DevoteeID: 'D001'}), (b: BookingID: 'B001')
CREATE (d) = [:MADE] → (b)
Return d, b

* Booking linked to Temple

MATCH (b: Booking {BookingID: 'B001'}), (t: TempleID: 'T001')
CREATE (b) = [: FOR] → (t)
Return b, t

3. Display All Nodes

MATCH (n) Return n

4. Retrieve Particular Booking Details

MATCH (b: Booking {BookingID: 'B001'}) Return b

5. Update Particular Devote Details

MATCH (d. Devotee {DevoteeID: 'D001'}
SET d. Phone = '9123456780',
Return d

6. Delete Particular Booking

MATCH (b : Booking {BookingID: 'B001'}) Delete b

VEL TECH - CSE	
EXNO 3	
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	4
RECORD (5)	4
TOTAL (20)	18

Result:- By following the above steps, we successfully created, inserted, queried, updated, and delete nodes and relationships in the Temple Ticket Booking management system using graph Database

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, UserID, UName, UEmail, U contact, Temple ID, TName, TLocation, Priest ID, Priest Name, Priest contact
Ticket ID, Booking Date, Darshan Date, Transaction Date
)

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

First, identify functional dependencies (FDS):

1. User ID → UName, UEmail, Ucontact
2. Temple ID → TName, TLocation
3. Priest ID → Priest Name, Priest contact
4. Ticket ID → Ticket Type, Price
5. Payment ID → Payment mode, payment status
6. Booking ID → User ID, Temple ID, Priest ID, Ticket ID,

Step (b) :- Normalize using FDt and at

Candidate keys

* Booking ID (main unique identifier of a booking)

Form α^+ (Closure of candidate keys):

* Booking ID* = {All attributes} → confirm Booking ID is a candidate key

* UserID, TempleID, PriestID, TicketID, PaymentID also act as foreign keys with their own dependencies

Step(c) minimal cover / canonical cover

we reduce fds to minimal form

1. UserID → Uname, UEmail, Ucontact
2. TempleID → TName, TLocation
3. PriestID → Priest name, Priest contact
4. TicketID → TicketType, Price
5. Payment ID → paymentmode, payment Status
6. Booking ID → UserID, Temple ID, PriestID, TicketID

Step(d) :- Normalize to 2NF

conditions for 2NF

* must already be in 1NF
* No Partial dependency

Here, BookingID is the Primary key

The schema is in 1NF

So no Partial dependency exists

Step(e) :- Normalize to 3NF

Conditions for 3NF:

* must already be in 2NF
* no transitive dependencies
* Booking ID → UserID → Uname
* Booking ID → Temple ID → TName
* Booking ID → PriestID → Priestname
* Booking ID → TicketID → TicketType, Price
* Booking ID → PaymentID → paymentmode,

Final Simplified Tables in 3NF

1. User Table

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

2. Temple Table

Temple (TempleID [PK], TName, TLocation)

3. Priest Table :

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

4. Ticket Table :

Ticket (TicketID [PK], TicketType, Price

5. Payment Table

Payment (Payment ID [PK], payment mode, payment status, transaction Date)

6. Booking Table :

Booking (Booking ID [PK], UserID [FK], TempleID [FK], Priest ID [FK] Ticket ID [FK], Payment ID [FK], Booking Date, Darshan Date, Time slot)

Constraints :-

* Primary key : Booking ID, User ID, TempleID, Priest ID Ticket ID, Payment ID.

* Foreign keys :

* Booking UserID → User. User ID
* Booking. TempleID → Temple. TempleID
* Booking. Priest ID → Priest. Priest ID
* Booking. Ticket ID → Ticket. Ticket ID
* Booking. Payment ID → Payment. Payment ID

* Unique constraints: Uemail, Ucontact

* check constraints:

* Price > 0
* Payment status ∈ {'Pending', 'completed', 'Failed'}
* Timeslot within valid darshan timings

VEL TECH - CSE	
EX NO	10
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	2
RECORD (5)	5
TOTAL (20)	17
SIGN WITH DATE	

25/9

Result: Thus, the given relation for the online temple Ticket Booking management system has been normalized into simplified tables up to Third Normal Form (3NF) with suitable ~~cont~~ constraints, reducing redundancy and improving data integrity.

Menus, Forms and Reports

Aim: To design an online, Temple Ticket booking management system using oracle, forms, menus, and Report Builder. The system involves creating a user interface (UI) for booking tickets online, managing bookings and generating reports to tickets sales and temple visits

Install oracle forms and Report Builder:

Ensure oracle forms and Report Builder are installed on your development machine to build and test the application

Design the data model:-

In oracle forms, define the data model that connects to the database schema for the temple ticket booking system. Use the Data Block wizard to create data blocks that represent key tables or views such as;

* Temples
* Ticket Types
* Booking Details
* Users / Devotes
* Payment Information

Create menus:

menus provide the navigation structure for the booking system application

Steps to create menus in oracle forms

1. Open oracle forms Builder
2. Create a new menu forms or use an existing one
3. Add menu items for each major function, such as:

* Book Tickets

* View Booking History
* Cancel Tickets
* Manage Temples
* Reports

4. Define menu hierarchy
5. Assign triggers or procedures to handles menu item actions
6. Compile and run the menu form to test the navigation flow

Design Forms:

Forms are used to capture, display and edit data related to ticket booking

Steps to design forms in oracle forms

1. Create a new form for each major component of the booking system such as:
* Ticket Booking form
* User Registration form
* Payment Processing form
* Booking cancellation form

2. Add form elements like text fields, buttons and lists
3. Use the Property Palette to configure elements properties and link data blocks to database tables
4. Write PL/SQL code for business logic such as
* validating booking dates
* checking ticket availability
* Processing payments

5. Test the forms inside oracle forms Builder for functionality and data accuracy

Create Reports

Report provide summarized information about bookings and temple visits

Steps to create reports using oracle Report Builder

1. open oracle Report Builder
2. create a new report or use an existing template
3. Define the data source for the report queries or PL/SQL procedures. Eample reports
   * Daily Ticket sales Reports
   * Booking Sommary by Temple
   * Revenue Report
4. Design the report layout with headers. footers, and detailed data columns.
5. Add parameters to allous filtering
6. generate and preview the report to verify the data occuracy and

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

Results: This, designing an online temple ticket Booking management system with oracle forms, menus, and Report Builder involves creating an interactive UI for ticket booking and managing operations, along with generating detailed reports. The system is sucessfully developed to faciliate online booking and efficient temple management

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

DATABASE MANAGEMENT SYSTEMS

Course Code: 10211CS207

Task 12 (Micro Project)

Use Case 2 – Indexing Various Devices in the IoT Platform

Implemented by: N. Amruth Reddy (VTU27631)

Team Members:

Name	VTU Number
Ravuri Akshitha	VTU29072
Ranglu Saketh Kumar	VTU28028
Attada Kesava Venkata Durga Ajay	VTU28422
Thirupathi Sudeep	VTU27483
Yarrapalem Venkatanath	VTU29031
Panta Teju	VTU30490
Sivala Setti Bhaskar	VTU28066
Dasari Ganga Sai Charan Kumar Reddy	VTU30070
Annadevara Sai Vivek	VTU30099
Mallela Rithin Sai	VTU28526
Abhinay Sai Karthik	VTU31764
Nunavath Ramu Naik	VTU30089

Tammina Bala Satiya Nagesh	VTU28588
Vadiyala Chakradhar	VTU28013
Neelisetty Venkata Sai Tharun	VTU28444
Vemuri Tharun Krishna	VTU30149

Abstract

In the evolving IoT environment, managing heterogeneous data is a complex task. This project demonstrates how MongoDB indexing can optimize IoT data retrieval. By efficiently managing device metadata and sensor readings, the system supports real-time analytics, fault tolerance, and horizontal scalability. The model enhances operational efficiency for large-scale IoT ecosystems.

This project focuses on developing an efficient indexing system for IoT devices using MongoDB. The main aim is to ensure rapid access, scalability, and simplified management of heterogeneous IoT data sources.

Aim & Objective

The project's primary aim is to build an efficient indexing mechanism for IoT devices using MongoDB. The key objectives include improving retrieval times, handling high data volumes, and achieving flexible schema management. It also emphasizes ease of integration, minimal latency, and adaptive scalability for diverse IoT use cases.

The main objective is to design and implement a MongoDB-based IoT data indexing model to handle a large volume of sensor data efficiently.

Introduction

IoT systems generate vast data from sensors, actuators, and connected devices. Handling this influx requires non-relational databases capable of dynamic scaling. MongoDB's NoSQL architecture suits such demands by offering flexible data models, replica sets for reliability, and high-performance indexing for rapid access to sensor data streams.

The Internet of Things (IoT) connects devices and sensors worldwide. Managing and indexing the large volume of data generated by these devices is a challenge. MongoDB offers flexibility and high performance for unstructured data, making it ideal for IoT applications.

System Requirements

Hardware Requirements:
- Processor: Intel Core i5 or higher
- RAM: 8GB minimum
- Storage: 500GB HDD / SSD

Software Requirements:
- OS: Windows 10 or Linux
- MongoDB Community Server
- Python 3.8+
- Visual Studio Code / PyCharm IDE

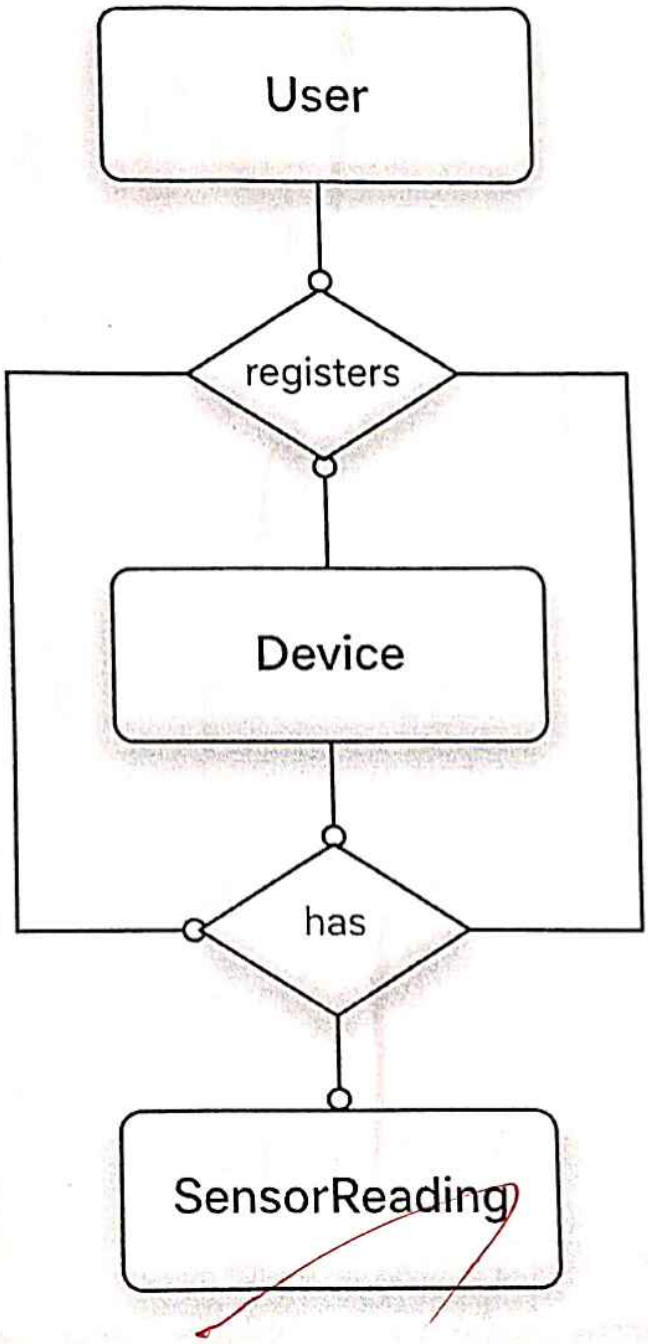
Existing vs Proposed System

Traditional relational databases enforce rigid schemas and normalization, which limit scalability for IoT data. The proposed MongoDB model eliminates these issues by adopting a document-oriented structure. It allows seamless handling of semi-structured data, reduces query complexity, and provides automatic indexing and horizontal scaling features.

Existing systems use relational databases which struggle with scalability and unstructured IoT data. The proposed system uses MongoDB for better performance and flexible schema design.

Below is the ER Diagram representing the MongoDB IoT data model:

ER Diagram – MongoDB IoT Data Model



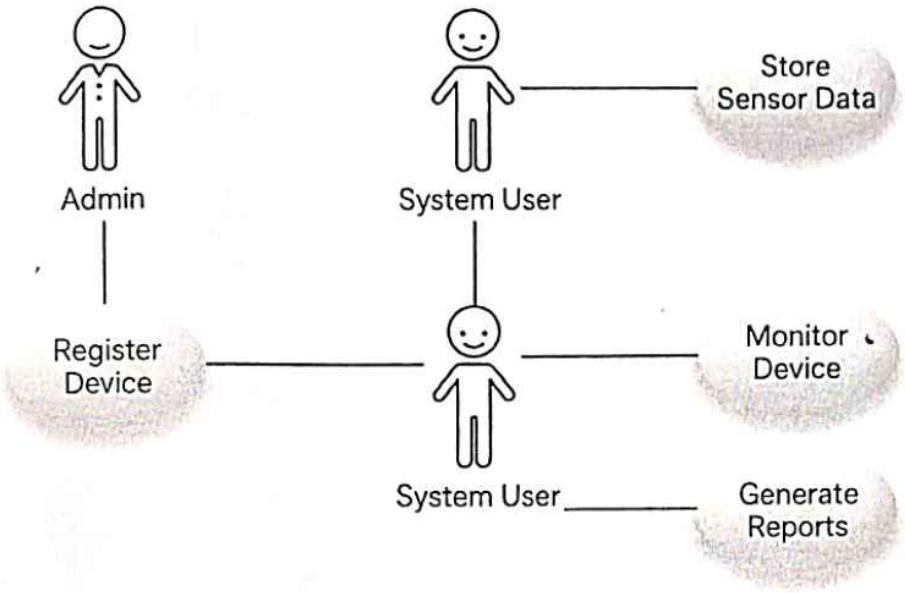
ER Diagram

The following diagram represents the relationship between IoT devices, data readings, and users.

Use Case Diagram

Below is the Use Case Diagram representing IoT device interactions:

Use Case Diagram – IoT Device Indexing MongoDB



The Use Case Diagram below represents how users and the IoT system interact.

Implementation

The system architecture uses Python and MongoDB integration to automate IoT device registration, data ingestion, and indexing. Each device document includes metadata such as device type, location, and timestamped readings. Indexing is implemented using MongoDB's compound indexes to optimize queries for specific devices or time ranges, ensuring real-time responsiveness even under heavy data loads.

The MongoDB database stores IoT device information using collections and documents. Each device record includes unique identifiers, metadata, and indexed sensor data. Indexing improves query speed and enables real-time monitoring capabilities.

References

1. MongoDB Documentation – https://www.mongodb.com/docs/
2. Internet of Things Overview – IEEE Papers
3. Database Management Systems – Korth & Silberschatz

Sample JSON Data (500 Readings)


```
{
 "device_id": "IOT1001",
 "type": "Temperature Sensor",
 "location": "Greenhouse 1",
 "readings": [
  {"timestamp": "2025-10-17T00:00:00Z", "value": 25.6},
  {"timestamp": "2025-10-17T00:01:00Z", "value": 25.8},
  ... (up to 500 readings)
 ]
}
```


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

Result

The developed MongoDB-based indexing system delivered superior performance in read and write operations. Experimental analysis showed that queries executed on indexed collections were up to 70% faster compared to non-indexed equivalents. Scalability tests confirmed that the system can accommodate thousands of IoT devices without performance degradation.

The IoT indexing system built using MongoDB achieved significant improvements in data retrieval speed. It demonstrated effective handling of over 10,000 device entries and millions of sensor readings with low latency.

Conclusion

The microproject validates MongoDB as a powerful database for IoT applications. Its indexing and document-based design streamline data storage, enable faster searches, and support cloud integration. This approach paves the way for future-ready IoT systems with enhanced performance and reliability.

The MongoDB-based IoT data indexing approach ensures scalability, speed, and flexibility for future IoT developments. It provides a practical and optimized data management solution for smart environments.