24/07/25

Task-1

A. Temple tickect online booking Mangment System

A temple tickect online booking management system allow devoteas to book tickects for temple visits special darshan, poojas, & other events online, reducing physical queoues & improving crowd managment. These system other include features like date & time slot selection, payment processing & the generation of tickects or passes

Entity:

The real-world object or concept that can be distinctly identital. Example includes students courses or product

~~set~~ Entity set:

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

Attributes:

A property or characteristic of an entity. for Example, a students might have attributs like name, ID & major.

Relationship:

An Association or interaction b/t two or more

dents might enroll

Task-1. conceptual Design after Fro

[Full Text Retrieval]

using basic datatype design methodology

ER Modeler, design, Entity Relationship

by satisfying the following sub tasks.

1.a Identifying the entities.

1.b: identifying the attributes.

1.c: Identification of relationships, cardinal

type of relationship

1.d: Reframing the relations with keys and

constrains-

cardinality:

1:N and 1:1 where Applicable

FTR: Full text Referevey

FTR: Full text Review

b/t the 'students' and 'course' entities.

Aim:

To design and develop an entity. relationship. (ER) Modes for a Temple ticket online Booking Management System that allows devotes to books tickets for temple visits. special darshans. Pooja, and other Events online, reducing Physical queues and improving crowd management.

Attributes:

* Devotes: Devotes - ID, Name, Phone, Email, address Age, Gender.

* Booking: Booking - ID, Date, Time - Slot, No. of persons, States.

* Events: Event - ID, Event - Name, Description, price, Event - type, duration.

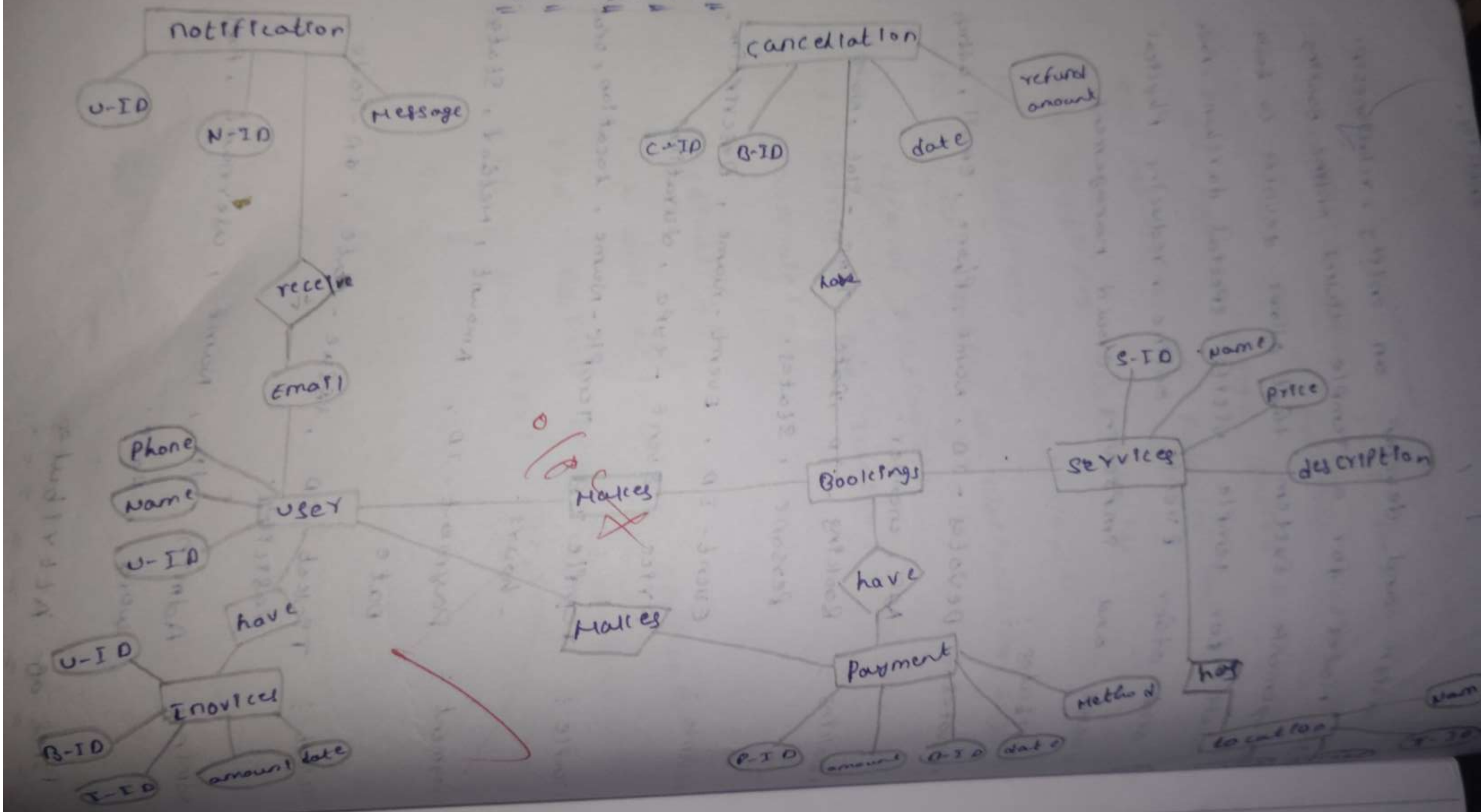
* Temple: Temple ID, Temple - Name, location, open - hours.

* Payment: Payment - ID, Amount, Method, States. Date

* Ticket: Ticket - ID, issue - Date, QR - code, validity.

* admin: Admin - ID, Name, username, Pass word, Role.

ER Diagram



notification
U-ID
N-ID
Message
receive
Email
User
Phone
Name
U-ID
have
Inovices
U-ID
B-ID
I-ID
amount
date
Makes
Makes
Cancellation
C-ID
B-ID
date
refund amount
have
Bookings
have
Payment
P-ID
amount
B-ID
date
Method
Services
S-ID
Name
Price
description
has
location

* multi : valued . Phone
* Derived ; Age (from DoB)

Relationships :

* Devotee -> Booking (Makes)
* Booking -> event (for)
* Booking -> Payment (has)
* Booking -> Tikect (generates)
* event -> Temple (host)
* 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 !

* I:N and 1:1 where applicable.

Result

The ER diagram for the temple Booking system was successfully designed shows all entities attributes & relationship with correct cordinallity for database implementations

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

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

Aim : Creating Hierarchical / Network model of the data base by enhancing the sound abstract data by Enhancing the sound obstract data by performing following task using forms of inheritance :

2.a Identify the specifiency of each relation ship, find a form surplus relations.

2.b . check is - a hierarchy / has - a - hierarchy. & perform generalization a/or specialization a/or relationship

2.c . Find the domain of the attribute a Perform check constraint of the applicable

2.d : Rename the relations.

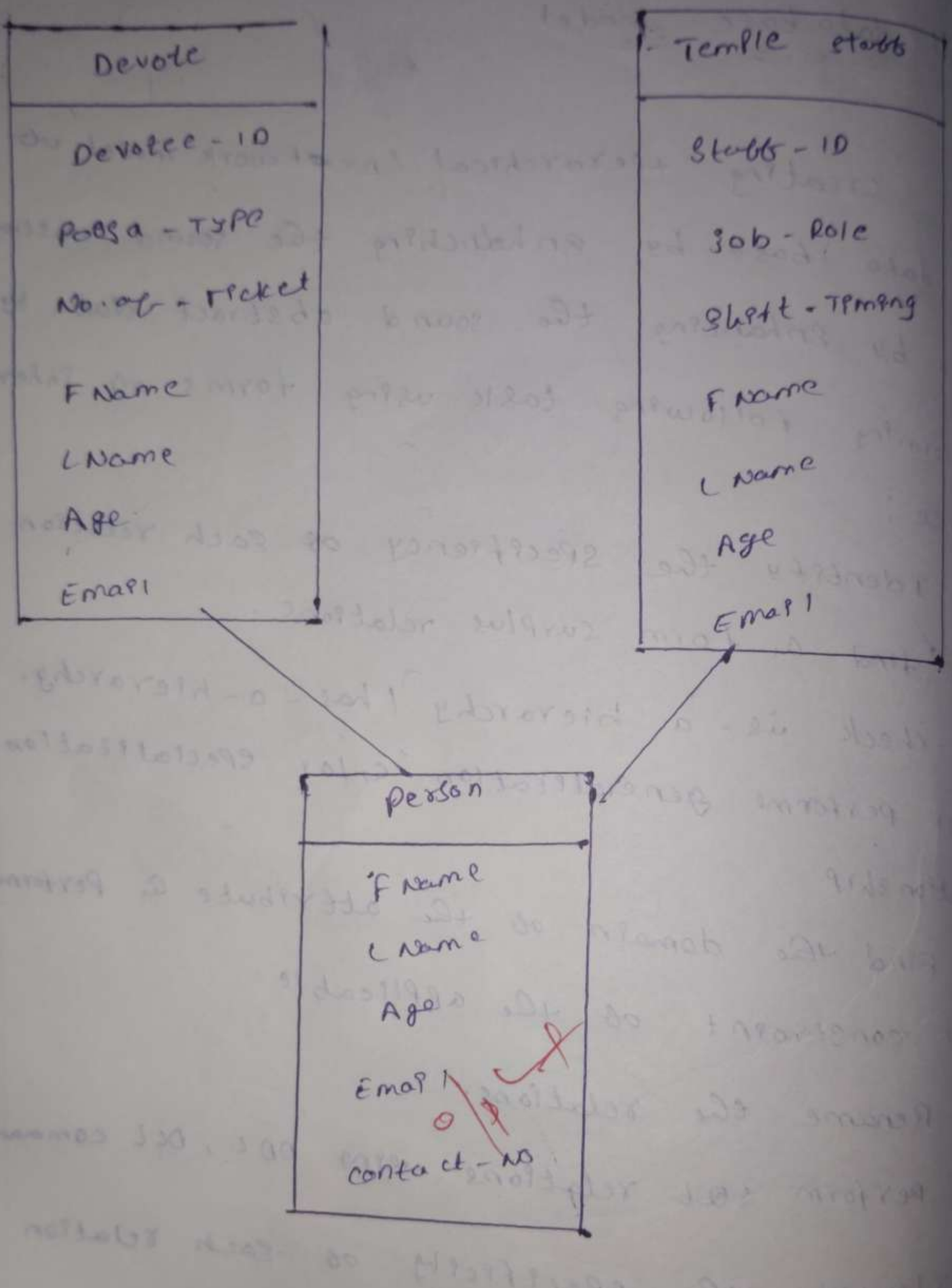
2.e : perform SQL relations using DDL, DCL commands

2a . Identify the specificity of Each relation ship, find a for surplus relations.

1. Relationship specificity :

a. one-to-many (Devotees ↔ Booking):

- Each devotee can have many bookings.
- Each booking is uniquely linked to one.



. one
. Ticl
c. on
. Ea
d. on
2. S
H
redun
izati
a. Ti
b. E
C. Po
2.b d
perfor
. devot
. even
serv
. Bookin
. Bookin
Gener
. Perso
. serv
spec

· Tickets is dependent on Booking.

c. one-to-one (Booking ↔ event

· Each booking is made for one event

d. one-to-one or one-to-many :

2. surplus relations ;

Here we identify entities that might be redundant, merged, or consolidated for normalization.

a. Ticket may be merged with booking.

b. Event details redundant in booking

c. Payment & Booking 1 : 1 relation.

2.b check is a hierarchy / has-a-hierarchy & perform generalization &/or specialization.

· devotee is a person.

· event / seva is a service of various temple service are modeled.

· Booking has-a ticket.

· Booking has-a relationship with user & temple

Generalization :

· Person (general) → devotee

· service → Event/seva

Specialization :

2c Attribute Domain & check constraints

- user. Phone - number : VARCHAR (15), check.
  LIKE [0-9] {10, 15 }.
- Booking .date - of - booking : DATE , Check. >=
  current - Data)
- Ticket .seat - count : Int : check (seat - count
  b/w 1 And 20)
- Event . type : ENUM (Darshan; Puja', 'abhishekam'),
  check (type IN (----))

2.d : Renaming Relations :

* user → Temple user

* Booking → Temple Booking.

* Ticket → Temple Ticket.

* Event → Temple Event.

2.e : SQL Relations using DDL, DCL

```
CREATE TABLE Temple user (
        user - id INT PRIMARY KEY
        name. VARCHAR (100) NOT NULL
        Phone . number VARCHAR (15) NOT NULL
Check (Phone - number - '^[0-9] {10, 15 } $')
    );

CREATE TABUE TEMPle (
  location VARCHAR (255)
 );
CREATE TABLE Temple Event (
      event-id INT PRIMARY KEY,
      temple-id INT REFERENCES
Temple (temple-id),
        event-type VARCHAR (20) CHECK,
(event-type IN ('Darshan', 'Pusa', 'Abhishekam')),
      event-date DATE,
      CONSTRINT chk-event-date CHECK
 (event-date >= CURRENT-DATE)
 );
CREATE TABIE Temple Booking (
          booking-id INT PRIMARY KEY,
          user-id INT REFERENCES
Temple user (user-id),
        event-id INT REFERENCES
Temple event (event-id),
          booking-date DATE DEFACULT
 CURRENT-DATE
 );
CREATG TABIE Temple Ticket (
          ticket-id INT PRIMARY KEY,
```

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)	18
SIGN WITH DATE	

Result:

Thus the Hierarchical model or Network model has been successfully created.

21/07

Date :- 7/08/25

Task - 3

using clouses, operators and function in queries

Aim :- To perform query processing on a Temple online Booking management system for different set visual result of queries using DML, DEL operations with aggregate functions, data function, string function. Set clockes, and operators.

Temple.

Temple ID	Name	Location	conTact - No
T1D01	Tirumala venkatesw-wore	Tirupati	9014276954
T1D02	meenakshi Amman	madurai	7989222925
T1D03	kashi vishwanath	varanasi	701392716
T1D04	Jaganath Temple	puri	9701224404
T1D05	Golden Temple	Amritsar	9908616499

visitor

visitoring	FName	LName	age	Email	contact.No
V001	Anil	kumar	30	anil@gmail .com	9014276954
V002	Akash	Reddy	25	Akash@gmail com	9701224404
V003	priya	Sharma	28	priya@gmail com	9908290010
V004	Aarav	Rates	25	Aaravp@gmail com	7013892716
V005	sneha	Rad	22	snehar@gmail com	7989222725

Booking

Booking ID	Temple ID	Visitor ID	Booking date	Ticket type	amount
B001	T1001	V001	2024-06-15	VIP	500
B002	T1002	V002	2024-06-16	General	100
B003	T1001	V003	2024-06-17	General	100
B004	T1003	V004	2024-06-18	General	700
B005	T1001	V005	2024-06-19	VIP	200

Priest

Priest ID	FName	LName	Age	Email	contact.No
P001	Ramesh	Iyer	50	ramesh@gmail com	9014276754
P002	Suresh	Sharma	45	suresh@gmail com	9701229404
P003	Manish	Das	40	Manish@gmail com	7989222785

2. Retrieve detailes of visitors where first name start with 'n'

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

Result;

Visitor ID	FName	LName	Age	contact.No
V002	Akash	Reddy	25	9701224404
V004	Aarav	Patel	25	7013982716

3. Add a colum for 'special, seva'. in Booking tables

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 'DS IT 001', 'TID02', and 'TID04',

```
SELECT *
FROM TEMPLE
WHERE TEMPLE IN ('TID01', 'TID03', TID04');
```

Result :
= = =

Temple ID	Name	location	contact No
TID01	Trimula	Tirupati	9701224464
TID02	kashi	Varanasi	7013982716
TID03	Golden temple	Amritsar	7989222725.

6. select visitor ID & names of visitor who booked special tickets.

```
Select visitor ID, FNAME, LName
FROM visitor
where visitor ID DNC
```

where Ticket ,type

Result :

```
visittion      FName      LName.
               sneha      Rao
V003
```

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

```
select Priest ID
  From Priest
where Temple ID is Null;
```

Result:

```
priest ID
(No Result if all priests are assigned).
```

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

7/8/25

Result : Thus. Query Processing for temple Ticket online Booking Management system using Clauses Opera tore, a functions was successfully performeN

Date : 14/7/25

Task 24:

using function in queries and writing sub queries

Aim: To perform advanced query processing and test its heristics by designing optimal correctated & nested subqueries such as finding summary statics in the temple ticket online Booking Managment System.

4.1 To retrieve all temple details, including the count of booking for each temple.

```
SELECT t.Temple ID,
        t.Name AS Temple Name
        t.location,
        t.contact-NO,
    COUNT (b.BookingID) AS Total Bookings.
FROM Temple t
LEFT JOIN Booking b
ON.t.Temple ID = b.Temple ID
GROUP BY t.Temple ID, t.Name, t.location, t.contact-No;
```

output:

Temple ID	Temple Name	location	Total Booking
TID01	Tirumala	Tirupati	3
TID02	Meenakshi Annam	Madurai	1
TID03	kashi	varanasi	1
TID04	Jagannath temple	puri	0
TID05	Golden temple	Amritsar	0

booking in a temple . wise manner.

```
SELECT t.Name AS Temple Name,
COUNT(*) AS Total special Bookings.
 FRONT Temple t
 JOIN Booking b
ON t. TEMPLEID = b. Temple ID
WHERE b. Ticket - type = 'special'
 GROUP BY t. Name;
```

output
= = =

```
Tample Name                          Total special Bookings

Tirumala Venkateshwara                      1
```

43: To Retrieve the details of temples where booking include 'VIP' tickets'

```
SELECT *
FRON Temple
WHERE Temple ID IN(
  SELECT Temple ID
  FROM Booking
WHERE Ticket - Type = 'VIP'
```

44: To retrieve visitor & booking details of visitor who are above 25 year old.

```
SELECT  v. Visitor . ID
        v. F Name AS Visitor Name.
          v. age .
          b. Booking ID.
```

wh

output :

temple ID	Name	Age	Booking [illegible]	Amount [illegible]
V00 1	Anil	20	B001	500
V00 3	Priya	28	B003	100
V00 4	Aaray	35	B004	300

4.5 : To retrieve 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
TI004	Jagannath Temple	Puri	9701224404
TI005	Golden temple	Amritsar	9014276954

4.6: To retrive the temple, Temple name, & Visitor name for a Particular visitorid given

```
SELECT t.temple ID,
       t.Name As Temple Name,
       V.F Name 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
T1001	Tirumala	Sneha

VEL TECH - CSE

EX NO	4
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	18
SIGN WITH DATE	

output :

Booking ID	devotee	Pooja Name	Booking amt
B101	Rasesh	Suprabhataseva	200
B102	meena	Archana	100
B103	Arsun	Abhishekam	500
B104	Jcogar	special	250
B105	Bharath	Darshan	300

output :

Devotees	Total Bookings
Rasesh	1
Arsun	1
Kartha	1
Suresh	1

28/08/25

Task No:- 5

17

Writing Join Queries, Equivalent, & /or recursive

Aim:- To perform advanced query processing & test heuristics using optimal correlation & nested subquery such as retrieving summary statistics & ticket booking details for the online Temple Ticket Booking management

5.1) To retrieve all temples & their available tickets.

```
select tm.Name AS Temple tk.tickets, tk.Type, TK
price
From Temple tm
Join Ticket tk ON tm.Temple ID = tk.Temple ID;
```

5.2) To list all booking along with Temple name & devotee name.

```
SELECT b.Booking ID, d.Devotee Name, tm.Name As Temple
Name, tk.Type, b.Booking Date

FROM Booking b
JOIN Devotee d ON b.Devotee ID = d.Devotee ID
JOIN Ticket tk ON b.Ticket ID = tk.Ticket ID
JOIN Temple tm ON tk.Temple ID = tm.temple ID;
```

5.3) To count number of booking made for each temple

```
As Temple Name
```

Output :

Devotee ID	FName	Mobile No
D103	Arsun	9876543211
D105	Suresh	8765321901

Output :

Poosa Name	Total Bookings
Suprabhata seva	1
Archana	1
Abhishekam	2
Special Darshan	1

```
LEFT 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 devotees, who booked tickets for. Tirupati Balaji

```
SELECT d. Devotee ID, d. Devotee Name, d. contact No,
  tm. Name AS Temple
FROM Devotee d
JOIN Booking b ON d. Devotee ID = b. Devotee ID
JOIN Ticket tk ON b. Ticket ID = tk. Ticket ID
JOIN Temple tm. ON tk. Temple ID = tm. Temple ID
WHERE tm. Name = 'Tirupati Balaji';
```

5.5) To retrieve all devotee details including total ticket booked'

```
SELECT d. Devotee ID, Devotee Name, d. contact No,
  COUNT (b. Booking ID) AS Total Tickets.

FROM Devotee d
LEFT JOIN Booking b ON d. Devotee ID = b. Devotee ID
GROUP BY. d. Devotee ID, d. Devotee Name, d. contact No
```

5.6) To retrieve the total number of special Darshan' ticket booked temple-wise output :-

Temple ID	Name	location	Head Prest
T001	Tirupati	Tirupati	Ramanujam
T002	kanchi	kachiPuram	Narayan Das
T003	Madurai	Madurai	Subramaniam

output :-

F Name	Age	Booking ID	Booking Date	Slottime status
Suresh	55	B105	24 JUN 2023	6:30 AM Confirmed

JO

W

W

5.7) TO

SELL

SELE

FRO

JOIN

WHE

5-8) T

above

SELECT

b . Bo

FROM

JOIN

JOIN

JOIN

WHER

5.9) TO

```
JOIN Booking b' ON tk. Ticket ID = b. Ticket ID
WHERE tk. Type = 'Special Darshan'
GROUP BY tm. Name;
```

5.7) To retrieve the temple details where tickets are still available

```
SELECT tm. Temple ID, tm. Name, tk. Type, tk. Price
FROM Temple tm.
JOIN Ticket tk ON tm. Temple ID = tk. Temple ID
WHERE tk. Available Seats > 0;
```

5.8) To retrieve devotee & Booking details for devotees above 40 years.

```
SELECT d. Devotee ID, d. Devotee Name, d. Age, b. Booking ID
b. Booking Date, tm. Name . AS Temple
FROM Devotee d
JOIN Booking b ON d. Devotee ID = b. Devotee ID
JOIN Ticket tk ON b. Ticket ID = tk. Ticket ID
JOIN Temple tm ON tk. Temple ID = tm. Temple ID
WHERE d. Age > 40;
```

5.9) To retrieve temple details where no bookings have been made

```
SELECT * FROM Temple
```

Output

Temple ID	Name	location	Head Offeg
T004	Ramesha Temple	Rameshwaram	Vishnu Devan.

Output

Temple	Poosa Name	Devotee	Booking Date	Slot tim
Tirupati	Archana	Meena	22 Jun 2023	9:00

5.10) To retrieve temple ticket ID, temple name & devotees name for a particular. 20

```
SELECT tm.temple ID, tk.Ticket ID, tm.Name AS
    Temple Name, d. Devotee Name
  FROM Temple tm
JOIN Ticket tk ON tm. temple ID = tk.Temple ID
JOIN Bookings b ON tk. Ticket ID = d.Devotee ID
WHERE b. Booking ID = 'B002';
```

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

Result: Thus the queries using Join Queries, Nested Queries & Equivalent Queries were executed successfully for the online temple ticket BMS.

18/09/25 Task : 6 : procedures, functions & Loops using PL/SQL

Aim : To write & executed PL/SQL Procedures functions, & Loops on Number. Theory & Business scenarios.

Algorithm :

(A) Number Theory Example - Prime Number check using Producere

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

B) Business scenario Example - Salary Bonus calculation using function

1. Start
2. create a function that accepts emp_salary.
3. if salary < 20000 -> 20% bonus .
   else -> 5% bonus
4. Return final salary with bonus.
5. End

Programs

(A) Number theory - Prime Number check

```
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;
/
----
```

(8) Business scenario - Employee Bonus calculation.

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
```

```
IF Salary < 20000 THEN
   bonus : = Salary * 0.20;
 ELSIF Salary B/W 20000 AND 50000 THEN
    bonus:= Salary * 0.10;
  END IF;
  RETURN (Salary + bonus.
 ;
 final -sal : = calc - bonus (20000);
  DBMS- OUTPUT. PUT -LINE ('Final Salary with Bouns:'
       || final- sal);

END;
/
```

Result: Thus, PL/SQL Procedures, functions, & loop were successfully implemented for number theory & business

EX NO	6
PERFORMANCE (5)	4
RESULT AND ANALYSIS (5)	4
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	17
SIGN WITH DATE	T.MiThr

VEL TECH - CSE

04-09-25

Task :- 7 Triggers, views & Exceptions.

24

Aim :-

To conduct events, views & exception handling on CRUD operation for the Temple Ticket online Booking Management system.

a) Trigger

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

```
CREATE OR REPLACE TRIGGER
trg_insert_ticket
AFTER INSERT ON Booking
FOR EACH Row
BEGIN
  INSERT INTO Ticket (Ticket ID, Booking ID, Devotees,
  Gender, QRcode, Status)
     values (
        seq_ticket.NEXTVAL,
      : New.Booking ID,
      'Unknown',
       NULL
       NULL,
      sys_GUID (),
      ACTIVE'
```

create a view to display booking details along with temple & slot information.

2.5

SQL

```
CREATE OR REPLACE VIEW Booking oetails view.
AS
SELECT
  b. Booking ID,
  b. Booking Ref.
  U. Name As Devotens
  t. Name As Temple
  dt. Name As Darshan Tune
  s. start TS AS slotstart,
  s. EndTS AS slotEnd,
  b. Qty,
  b. Amount,
  b. Status.
FROM Booking b
JOIN users U ON b. userID = U. userID
JOIN Temple t ON b. Temple D = t. Temple ID
dt. Darshan Type ID ;
SELECT * FROM Booking Details view ;
```

c) Non - Recursive PL/SQL Procedure

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

```
out-even-bookIng-Ids OUT
sys.ODCINUMBERLIST
)AS
BEGIN
  out-even-bookIng-Ide :=
  sys.ODCINUMBERLIST();
  for rec IN (
     SELECT BookIng ID
     FROM BookIng
   WHERE TempIe ID = In-TempIe-Id
   AND MOD (BookIng ID, 2) =0
  ) LOOP
   out-even- bookIng -Ids. EXTEND;
out-even-bookIng-Ids) := rec.BookIng ID;

  END LOOP;

 END;
/
```

Execution Block;

Soil

```
DECLARE
  temple-id  Num := 101;
  even-bookIng-Ids sys. ODCINUMBERLIST;
BEGIN
  Get EvenBookIng IDs for TempIe (temple-Id, even-Ids)
                                    COUNT .LOOP
```

output

SELECT FROM * Booking

Booking ID	DevoteeID	Temple	Booking ID
102	1002	202	04-sep-25
101	1001	201	04-sep-25

SELECT * FROM Booking_status;

STATUS ID	Booking ID	DEVOTEE ID	STATUS
41	102	1002	Pending
61	101	1001	Pending

```
END
temple_id NUMBER
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;
/
```

Result: Thus the triggres, views & & exceptions for the System were successfully implemently & verified.

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	04.05.

11/09/25

Task : 8 : CRUD operations in Document database.

Aim : To design & interact with a mango DB. document database using & NPM, & to Perform basic CRUD operation

Algorithm:

1. Install dependinrgcies Intialize with a mango via Node is & NPM, & to to Perform basic CRUD operation

2. connect to mango DB : use Mongoose to connect to a local or cloud Mongo DB database.

3. Define a schema : create a Mangoose schema, for the docuement

4. create a model : Based on the define a model to interact with the collection.

5. Perform CRUD operations:
   - create / Inset : add new decuements.
   - Read / find Existing docuements dased on dibry
   - updata : Modify existing docuements.
   - Delete / Remove : Remove docuements from the collection

1. Initialize Project & install Mangooge .is.

```
const mongoose = reaupres ('Mongoose);
// step 1: Connect to Mongo DB
Mongoose Connect (
console . log C" connected to mongo DB");
}). catch  cerror C" MongoDB connection error; err);
}) ;
// step 2 : Define a schema
const ticket () schema = new mongoose . schema ({
   name : string .
  visit : string.
  ticket type : Number
}) ;
// step 3 : create a model.
 const user = mongoose . model ('user', user schema) ;
// CREATE : Insert a new user.
 const create = async () => {
  const user = new user ({
   name : " Alice",
  ticket type : " VIP"
}) ;
const get user = async (x-x {
  const user = await user
 await ticket . save () ;
 console . log (" Ticket created! ", ticket );
};
  // Read
```

Output

```
Ticket created : {
  _id : "650123 abc'",
  Name : "Sita Devi";
  visit Data : "2025-09-15",
  ticket Type : "VIP",
  __v:0
}.
```

O/P

```
const upda
const update = await Ticket find By Id
And update (id) await Ticket Type : "General"},
    { new! true });
  console.log ("update ticket :", update);

};
 // DELETE
const delete Ticket = async (id) => {
  await Ticket .find By (Id And Delete (id);
  console.log " (Ticket Deleted :)", id ) ;

};
```

How to Run

1. Start Mongo DB (Local)
2. In terminal.

```
node app.js
```

VEL TECH - CSE	
EX NO	8
PERFORMANCE (5)	4
RESULT AND ANALYSIS (5)	4
VIVA VOCE (5)	4
RECORD (5)	x
TOTAL (20)	12
SIGN WITH DATE	Divya 11.09.28

Result : Thus the program to to design interact with mongo DB decuement database using Mangoose via

18/09/25   Task-9

CRUD operations on Graph Database

Aim: To design a graph database model for tempel ticket booking using a Graph & Perform CRUD opera tions.

Algorithm:

1. Define graph entities - temple devotes & ticket
2. Define relationships - userbooked - for ticket for temple
3. Design Graph QL schema types for there entity with relationship
4. use cypher queries for CRUD operation
   - * Creat Nodes
   - * Insert nodes
   - * Query graph
   - * Delete Nodes.
5. Expose a Graph QL API using Neon j.graph. Integration for client side CRUD operation Int raction

Source code:

```
Type temple {
    Templeid : ID,
      name : string;
      location : string
```

output
= : :

```
{
 " name "  , " Ravi'' ;
 " bookings  ; [ {
  " ticket Id" ; " 501 "
  " temple " ;  { 'name " . Tirupati " . } /.
  " Pooja " : { " name " ; 'abhishekam" }
```

8/10

data string.

seat number ; string

status : string.

user : users ; { relationship p ctype

" Booked : for ", direction = (N)

temple ; Temple ! @ relationship (type :

" for temple ". direction }

creat Nodes,

CREATE (t . temple { templeid'' , Name : 's

temple ; location : 'city A' });

CREATE (u : User { user ID : '01' . name : 'Alice',

email : alice @ example . com' })] :

CREATE (u : ticket { ticket ID : 'tk' , date

2025 - 10 - 01 : seat Number = 'A10' status -

(Booked ' }) !

find ticket for user ;

MATCH (" ; user [user id : (u'; }] (Booked = for)

-> [tk . ticket]

Return (t.k ticket ; d , tk . data , tk . seat

number t.k status) ;

update ticket status

MAtch

(tk . ticket [illegible] (TK , })

Delete ticket

```
MATCH (tk:ticket {ticket ID: 'TK1'})
      DETACH DELETE tk;
```

VEL TECH - CSE	
EX NO	9
PERFORMANCE (5)	4
RESULT AND ANALYSIS (5)	4
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	17
SIGN WITH DATE	G. Mathy 18.09.24

Result: Thus the Program to design the graph space & CRUD operation is verified & successfully completed.

25/09/24
Task : 10

Normalizing database using functional dependenci up to third normal form.

Aim : To Normalize the below relation & create the simplified folder 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. T Name, T location, Priest ID, Priest Name, Priest contact, Ticket ID, Booking payment ID, Payment mode, Payment - status Transation Date
).

Step (a): Apply function dependencies → Normalize of 1NF

1. user ID → UName, U Email, U contact.
2. Temple ID → T Name. Contact.
3. Priest ID → Priest Name, Priest contact,
4. Payment ID → payment mode, payment status. Tranaction date
5. ticket ID → Ticket type, price

Step (b) : Normalize using FD & at condidate key:

* Bookin ... at a Bookin

From at (clouser ob candidate keys):

- Booking ID = { All Attributes)
- user ID, Temple ID, Priest ID, Ticket payment ID also at of foreign keys.

step(c): minimal || canonical covers we reduce FDs to minimal form.

1. user ID
2. Temple ID
3. Priest ID
4. Ticket ID
5. Payment ID
6. Booking ID

* Already minimal & canonical -> no redudant attributes.

step(d):

Normalize to 2NF

conditions for 2NF

* must be alredy be in 1NF
* No Partial dependency.

Here. Booking ID, is the Priccary key so no Partial dependency Exists.

the schema is in 2NF

step(e): Normalize to 2NF

* Must
* No transitive

check:

* Book ID → user ID → Uname
* Book ID → temple ID → Tname
* Booking ID → priest ID → priest Name
* Booking ID → Ticket ID → Ticket type, price
* Booking ID → Payment status.

Final simplified tables in 3NF

1. user table.

user (user ID (PK), Uname, U Email, U contact)

2. temple table.

temple (temple ID (PK), Tname, T location)

3. priest table.

priest (priest ID (PK), priest name, priest contact)

unique constraint:

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

VEL TECH - CSE	
EX NO	10
PERFORMANCE (5)	4
RESULT AND ANALYSIS (5)	4
VIVA VOCE (5)	4
RECORD (5)	5
TOTAL (20)	17
SIGN WITH DATE	

Result: Thus the given reason for the temple booking management has been normalized sample tables to 3NF with suitable constraint

09/10/ Task 11 menus, forms & Reports

Aim: To design an online temple ticket booking management system using oracle forms, menu & report Builder. The system involves creating reports related to ticket sales & tomple visits.

Install orcale forms & report Builder.

Ensure oracle forms & report Builder are installed on your development machine to build & test application.

Design the Date model:

In oracle mode, define the data that connected to d.b schema.

- Temple
- Ticket type
- Booking details
- user.
- payment information.

create menus!

menus provide the navigation structure for booking System application

Steps to create menus in oracle turn:

1. open application from building
2. Create a new menu from or use an existing one
3. Add menu items for each major function such as!
   - book tickets

· cancel tickets

· Reports

4). define menu Hierarchy

5) Assign triggers or procedures to handle menuitem

6) compile & run the menu form.

design forms:-

forms are used to capture, display & editdata

steps to design forms in oracle forms

create a new form for each major component.

of booking system such as,

· Ticket booking form

· Payment proccessing form.

· Add form element like text fields. bottons & lists.

· use property plate to configure element proper

· test forms inside oracle forms builder for

function & data accuracy.

create reports

Reports Provide sommariced information about booking & temple visits.

steps to create reports using oracle report

1. open oracle report builder.

2. create a new report or use an temparate

3. define the data source for report using queries or PL·SQL procedures.

4. Add Parameters to allow filtering.

5. generate a Preview the result to very the data accuracy and Presentation.

VEL TECH - CSE	
EX NO	9
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	✓
TOTAL (20)	20
SIGN WITH DATE	09.10.24

Result : Thus designing an online ticket booking system with oracle forms menus a reports.

operations.

16/10/2025

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

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

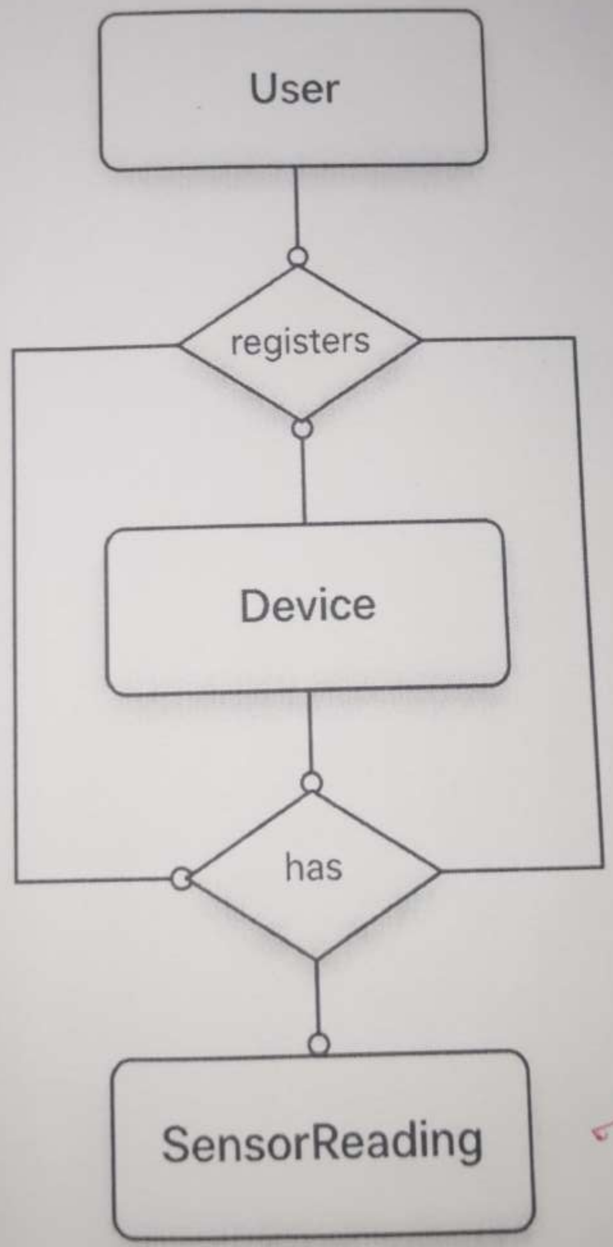
Existing vs Proposed System

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

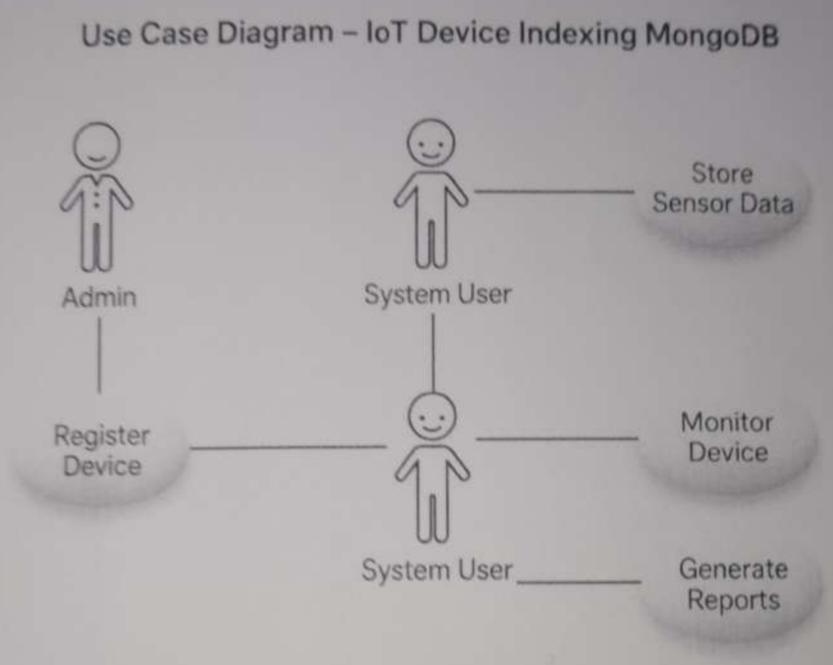


ER Diagram

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.

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)
 ]
}
```

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.

16/10

VI TECH - CSE

EX NO	UC
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	3
RECORD (5)	5
TOTAL (20)	18
SIGNATURE	