1pr 1.

1.Create a table named Employees with the following columns:

· employee_id (primary key, auto-increment)

· first_name (not null)

· last_name (not null)

· email (unique, not null)

· hire_date (default to the current date)

· salary (greater than or equal to 3000)

2. Create a sequence employee_seq that starts at 1001 and increments by 1. Use this sequence to insert a new employee record into the Employees table.

3. Create a view EmployeeView that shows the employee_id, first_name, last_name, and salary of employees who have a salary greater than or equal to 5000.

4. Create an index on the email column of the Employees table to speed up search queries based on email addresses.

5. Create a synonym Emp for the Employees table to simplify referencing the table in future queries.

6. Add a new table Departments with the columns:

· department_id (primary key)

· department_name (not null) Then, alter the Employees table to add a foreign key constraint that references department_id from the Departments table.

```sql
CREATE TABLE Employees (
    employee_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    hire_date DATE DEFAULT CURRENT_DATE,
    salary DECIMAL(10, 2) CHECK (salary >= 3000),
    department_id INT
);

INSERT INTO Employees (first_name, last_name, email, salary)
VALUES ('Rahul', 'Sharma', 'rahul.sharma@indianmail.com', 5500);

CREATE VIEW EmployeeView AS
SELECT employee_id, first_name, last_name, salary
FROM Employees
```

```sql
WHERE salary >= 5000;

CREATE INDEX idx_email ON Employees (email);

CREATE TABLE Departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL
);

ALTER TABLE Employees
ADD CONSTRAINT fk_department
FOREIGN KEY (department_id) REFERENCES Departments(department_id);

INSERT INTO Departments (department_id, department_name)
VALUES (1, 'Engineering'), (2, 'HR'), (3, 'Marketing');

INSERT INTO Employees (first_name, last_name, email, salary, department_id)
VALUES ('Priya', 'Iyer', 'priya.iyer@indianmail.com', 7000, 1);

INSERT INTO Employees (first_name, last_name, email, salary, department_id)
VALUES ('Amit', 'Patel', 'amit.patel@indianmail.com', 8000, 2);

CREATE SEQUENCE employee_seq
START WITH 1001
INCREMENT BY 1;

INSERT INTO Employees (employee_id, first_name, last_name, email, salary)
VALUES (NEXTVAL('employee_seq'), 'John', 'Doe', 'john.doe@indianmail.com',
6000);

CREATE SYNONYM Emp FOR Employees;
```

2pr

1. Create a table named Employees with the following columns:

· employee_id (primary key, auto-increment)

· first_name (not null)

· last_name (not null)

· email (unique, not null)

· hire_date (default to the current date)

· salary (greater than or equal to 3000)

2. Alter the salary column in the Employees table to increase its size to DECIMAL(15, 2). 3. Rename the Employees table to Staff.

4. Create a composite index idx_name_salary on the first_name and salary columns in the Employees table to speed up queries that filter on both fields.

5. Create a check constraint on the hire_date column of the Employees table to ensure that no employee can have a hire date in the future.

```
CREATE TABLE IF NOT EXISTS Employees (

    employee_id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    hire_date DATE DEFAULT CURRENT_DATE,
    salary DECIMAL(10, 2) CHECK (salary >= 3000)
);

CREATE SEQUENCE IF NOT EXISTS employee_seq
START WITH 1001
INCREMENT BY 1;

INSERT INTO Employees (employee_id, first_name, last_name, email, salary)
VALUES (NEXTVAL('employee_seq'), 'Rahul', 'Sharma',
'rahul.sharma@indianmail.com', 5500);

CREATE VIEW EmployeeView AS
SELECT employee_id, first_name, last_name, salary
FROM Employees
WHERE salary >= 5000;

CREATE INDEX IF NOT EXISTS idx_email ON Employees (email);

CREATE TABLE IF NOT EXISTS Departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL
);

ALTER TABLE Employees
ADD CONSTRAINT fk_department
FOREIGN KEY (department_id) REFERENCES Departments(department_id);

INSERT INTO Departments (department_id, department_name)
VALUES (1, 'Engineering'), (2, 'HR'), (3, 'Marketing');

INSERT INTO Employees (employee_id, first_name, last_name, email, salary,
department_id)
VALUES (NEXTVAL('employee_seq'), 'Priya', 'Iyer', 'priya.iyer@indianmail.com',
7000, 1);

INSERT INTO Employees (employee_id, first_name, last_name, email, salary,
department_id)
```

```
VALUES (NEXTVAL('employee_seq'), 'Amit', 'Patel', 'amit.patel@indianmail.com',
8000, 2);

ALTER TABLE Employees
MODIFY COLUMN salary DECIMAL(15, 2);

RENAME TABLE Employees TO Staff;

CREATE INDEX IF NOT EXISTS idx_name_salary ON Staff (first_name, salary);

ALTER TABLE Staff
ADD CONSTRAINT chk_hire_date CHECK (hire_date <= CURRENT_DATE);
```

3 PR

1. Create the following tables :

· **Students** (student_id, first_name, last_name, email, department_id, enrollment_date, phone_number)

· **Departments** (department_id, department_name)

· **Courses** (course_id, course_name, department_id, course_fee)

· **Enrollments** (enrollment_id, student_id, course_id, enrollment_date, grade)

2. Get a list of all students in the university, showing their first and last names, email, and department name.

3. Insert a new student into the Students table.

4. Update the email address of the student with student_id = 1002.

5. Delete a student record from the Students table where student_id = 1002. 6. Retrieve all courses offered by the "Computer Science" department, including the course name and fee.

7. Find the names of all students enrolled in the course "Database Systems." 8. Insert a new enrollment for student student_id = 1001 in the course course_id = 201, setting the enrollment date to today's date and giving the grade 'A'.

```
-- Create the 'department' table first, as we need this table for foreign key
reference
```

```sql
CREATE TABLE department (
deptid INT PRIMARY KEY,
dpname VARCHAR(40)
);

-- Create the 'student' table
 -- Changed phone to VARCHAR to accommodate phone numbers
-- Foreign key to 'department'
CREATE TABLE student (     id
INT PRIMARY KEY,     fname
VARCHAR(50),     lname
VARCHAR(50),     email
VARCHAR(50),     did INT,
edate DATE,
    phone VARCHAR(15),
    CONSTRAINT fkdp FOREIGN KEY (did) REFERENCES department (deptid)
);




-- Create the 'course' table with the foreign key reference to 'department'
 -- Adding the primary key constraint for 'cid'
CREATE TABLE course (     cid INT,     cname
VARCHAR(50),     deparid INT,     course_fee
INT,
    FOREIGN KEY (deparid) REFERENCES department (deptid),
    PRIMARY KEY (cid)
);

-- Create the 'enrollments' table
CREATE TABLE enrollments (
eid INT,     studid INT,
course_id INT,     endate DATE,
grade VARCHAR(10),
    FOREIGN KEY (studid) REFERENCES student(id),
    FOREIGN KEY (course_id) REFERENCES course(cid)
);

-- Insert data into the 'department' table
INSERT INTO department (deptid, dpname)
VALUES
    (101, 'Computer Science'),
```

```
    (102, 'Mathematics');

-- Insert data into the 'student' table
INSERT INTO student (id, fname, lname, email, did, edate, phone)
VALUES
    (1, 'John', 'Doe', 'john.doe@example.com', 101, '2024-08-01', '1224'),
    (2, 'Jane', 'Smith', 'jane.smith@example.com', 102, '2024-08-15', '9234'),
    (3, 'Alice', 'Johnson', 'alice.johnson@example.com', 101, '2024-09-01',
'55567'),
    (4, 'Bob', 'White', 'bob.white@example.com', 102, '2024-09-10', '555543');
-- Select query to join 'student' and 'department' tables and show results
SELECT fname, lname, email, dpname
FROM student
JOIN department ON student.did = department.deptid;

--update and add studnet   insert into department values(201,"AIDS")  insert
into student values(1001,"swaraj","gaikwad","abc@gamil.com",201,"2004-
11-11",12345);  update student set
email="swaraj@gmail.com",id=2001 where did=201;
 --delete
 DELETE FROM student WHERE id = 1002;
 --insert two course    insert into course
values(1,"OOP",101,50000),(2,"DSA",101,30000);
--enter in enrol  insert into enrollments
values(01,1,3,"2024-11-12","A);
```

4PR

same from above

```
update enrollments set grade="B+" where studid=2001;

--get count of cousre
SELECT course.cname, COUNT(enrollments.studid) AS num_students
FROM course
LEFT JOIN enrollments ON course.cid = enrollments.course_id
GROUP BY course.cname;

--no stud enroll
 1)select * from student left join enrollments on student.id =
enrollments.studid where enrollments.studid is null;
```

```
 2)select * from student where student.id not in (select enrollments.studid
from enrollments);

--no mathematics  select * from student where did=(select deptid
from department where dpname="Mathematics");
```

5pr

Design and Develop MongoDB Queries using CRUD operations

1.Insert a document into the products collection with the following data:

· name: "Laptop"

· category: "Electronics"

· price: 1200

· stock: 50

2. Insert multiple documents into the orders collection. Each document should have the following structure:

· orderId: Unique order identifier

· customerName: The customer's name

· items: Array of products (each product has name and quantity)

· orderDate: Date of the order

3. Query the products collection to find all documents where the category is "Electronics".

4. Query the products collection to find all products where the price is greater than 1000.

5. Update the stock quantity of the product with name "Laptop" in the products collection to 45.

6. Update the stock of all products in the Electronics category by adding 10 to the existing stock. 7. Delete the order with orderId 101 from the orders collection.

```
--insert many db.products.insertMany([  { oid: 1001, cname:"abc",  items: [{
name: "phone" },  { name: "laptop" }],
 odate: new Date("2024-11-11")
 },
 { oid: 1002, cname: "xyz",   items: [{ name: "iphone" },
 { name: "tab" }],
```

```
 odate: new Date("2024-11-12") }])

//price   db.products.find({ price: { $gt:
1000 } });
//update  db.products.updateMany(
    { category: "Electronics" },
    { $inc: { stock: 10 } }
);
```

6PR

6.

Design and Develop MongoDB Queries using CRUD operations

1. Insert a document into the users collection with the following data:

· name: "Sara Lee"

· email: "sara.lee@example.com"

· age: 28

· address: { city: "San Francisco", state: "CA" }

2. Insert multiple blog post documents into the posts collection. Each document should have: · title: Post title

· author: Author's name

· content: Blog content

· tags: Array of tags

· published: Boolean indicating if the post is published

3. Query the users collection to find all users where age is greater than 30. 4. Query the posts collection to find all blog posts that are marked as published: true. 5. Update the address of the user with name: "Sara Lee" to the following: · city: "Los Angeles" · state: "CA"

6.Add a new tag "Tutorial" to the post with title: "Introduction to MongoDB" in the posts collection.

```
db.users.insertOne({     name:
"Sara Lee",     email:
"sara.lee@example.com",     age:
28,
```

```
    address: {         city:
"San Francisco",
state: "CA"
    }
});

//for post
 db.posts.insertMany([
    {         title: "Introduction to MongoDB",         author:
"John Doe",         content: "This post introduces MongoDB, a NoSQL
database.",         tags: ["mongodb", "database", "NoSQL"],
published: true
    }
]);
 //push
db.posts.updateOne({title:"tot"},{$push:{tags:"tutorial"}})
```

7pr

7. Unnamed PL/SQLcode block: Use of Control structure and Exception handling is mandatory.

Suggested Problem statement:

Consider Tables:

1. Borrower (Roll_no, Name, Date_of_Issue, Name_of_Book, Status)

2. Fine (Roll_no, Date, Amt)

· Accept Roll_no and Name_of_Book from user.

· Check the number of days (from Date_of_Issue).

· If days are between 15 to 30 then fine amount will be Rs 5per day.

· If no. of days>30, per day fine will be Rs 50 per day and for days less than 30, Rs. 5 per day.

· After submitting the book, status will change from I to R.

· If condition of fine is true, then details will be stored into fine table.

· Also handles the exception by named exception handler or user define exception handler.

8pr

8. **Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor)** Write a PL/SQL block of code using parameterized Cursor that will merge the data available in the newly created table N_Roll_Call with the data available in the table O_Roll_Call. If the data in the first table already exists in the second table then that data should be skipped.

```
-- Step 1: Create tables
CREATE TABLE N_Roll_Call (
id INT PRIMARY KEY,
student_name VARCHAR(255),
roll_no INT
);

CREATE TABLE O_Roll_Call (
id INT PRIMARY KEY,
student_name VARCHAR(255),
roll_no INT
);

-- Step 2: Insert sample data into N_Roll_Call
INSERT INTO N_Roll_Call (id, student_name, roll_no) VALUES
(1, 'Alice', 101),
(2, 'Bob', 102),
(3, 'Charlie', 103),
(4, 'David', 104);

-- Step 3: Create stored procedure
DELIMITER $$

CREATE PROCEDURE MergeRollCallData()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE v_id INT;
    DECLARE v_student_name VARCHAR(255);
    DECLARE v_roll_no INT;
    DECLARE cur CURSOR FOR
        SELECT id, student_name, roll_no FROM N_Roll_Call;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;
    OPEN cur;

    read_loop: LOOP
        FETCH cur INTO v_id, v_student_name, v_roll_no;
```

```
        IF done THEN
LEAVE read_loop;
        END IF;

        INSERT IGNORE INTO O_Roll_Call (id, student_name, roll_no)
        VALUES (v_id, v_student_name, v_roll_no);
    END LOOP;

    CLOSE cur;
END$$

DELIMITER ;

-- Step 4: Call the procedure to merge data
CALL MergeRollCallData();

-- Step 5: Check the result
SELECT * FROM O_Roll_Call;
```

Pr9

9.

1.Create the students table with the following columns:

· student_id (Primary Key, Integer, Auto-increment)

· first_name (Varchar, 100 characters)

· last_name (Varchar, 100 characters)

· birthdate (Date)

· email (Varchar, 100 characters, Unique)

· enrollment_date (Date)

2. Add a column phone_number (Varchar, 15 characters) to the students table. 3. Change the data type of the email column from VARCHAR(100) to VARCHAR(255). 4. Remove the phone_number column from the students table.

5. Rename the enrollment_date column to date_of_enrollment in the students table. 6. Create an index on the email column to speed up queries that filter or search by email. 7. Drop the entire students table from the database.

```
-- Step 1: Create the students table with the specified columns
CREATE TABLE students (     student_id INT AUTO_INCREMENT
PRIMARY KEY,     first_name VARCHAR(100),
```

```
    last_name VARCHAR(100),
birthdate DATE,     email
VARCHAR(100) UNIQUE,
enrollment_date DATE
);

-- Step 2: Add a column 'phone_number' to the students table
ALTER TABLE students
ADD COLUMN phone_number VARCHAR(15);

-- Step 3: Change the data type of the email column from VARCHAR(100) to
VARCHAR(255)
ALTER TABLE students
MODIFY COLUMN email VARCHAR(255);

-- Step 4: Remove the 'phone_number' column from the students table
ALTER TABLE students
DROP COLUMN phone_number;

-- Step 5: Rename the 'enrollment_date' column to 'date_of_enrollment'
ALTER TABLE students
RENAME COLUMN enrollment_date TO date_of_enrollment;
-- Step 6: Create an index on the email column to speed up queries filtering
by email
CREATE INDEX idx_email ON students (email);
-- Step 7: Drop the entire students table from the database
DROP TABLE students;
```

Pr 10

10.

1. Create the students table with the following columns:

· student_id (Primary Key, Integer, Auto-increment)

· first_name (Varchar, 100 characters)

· last_name (Varchar, 100 characters)

· email (Varchar, 100 characters)

· birthdate (Date)

2. Insert a new student into the students table with the following information: · first_name: John

· last_name: Doe

· email: john.doe@example.com

· birthdate: 1998-05-12

3. Insert two new students into the students table:

1. first_name: Jane, last_name: Smith, email: jane.smith@example.com, birthdate: 2000-08- 22

2. first_name: Alice, last_name: Brown, email: alice.brown@example.com, birthdate: 1999- 11-15

4. Select all columns from the students table to display all student records. 5. Retrieve the first_name, last_name, and email of all students.

6. Update the email address of the student with student_id = 1 to john.doe2024@example.com. 7. Delete the student record where student_id = 3.

```
-- Step 1: Create the students table with the specified
columns CREATE TABLE students (     student_id INT
AUTO_INCREMENT PRIMARY KEY,     first_name VARCHAR(100),
last_name VARCHAR(100),     email VARCHAR(100),     birthdate
DATE
);

-- Step 2: Insert a new student into the students table
INSERT INTO students (first_name, last_name, email, birthdate)
VALUES ('John', 'Doe', 'john.doe@example.com', '1998-05-12');
-- Step 3: Insert two new students into the students table
INSERT INTO students (first_name, last_name, email, birthdate)
VALUES
('Jane', 'Smith', 'jane.smith@example.com', '2000-08-22'),
('Alice', 'Brown', 'alice.brown@example.com', '1999-11-15');
-- Step 4: Select all columns from the students table to display all student
records
SELECT * FROM students;

-- Step 5: Retrieve the first_name, last_name, and email of all students
SELECT first_name, last_name, email FROM students;

-- Step 6: Update the email address of the student with student_id = 1
UPDATE students



SET email = 'john.doe2024@example.com'
```

```sql
WHERE student_id = 1;

-- Step 7: Delete the student record where student_id = 3
DELETE FROM students WHERE student_id = 3;
```