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)

Ans: CREATE TABLE Employees (

employee_id INT PRIMARY KEY AUTO_INCREMENT,

first_name VARCHAR(50) NOT NULL,

last_name VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

hire_date DATE DEFAULT CURRENT_DATE,

salary DECIMAL(10, 2) CHECK (salary >= 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.

-- Create the sequence

CREATE SEQUENCE employee_seq

START WITH 1001

INCREMENT BY 1;

-- Insert a new employee using the sequence

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

VALUES (employee_seq.NEXTVAL, 'John', 'Doe', 'john.doe@example.com', 5000);

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.

Ans:

```
CREATE VIEW EmployeeView AS

SELECT employee_id, first_name, last_name, salary

FROM Employees

WHERE salary >= 5000;
```

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

```
CREATE INDEX idx_email ON Employees (email);
```

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

Ans: CREATE SYNONYM Emp FOR Employees;

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.

Ans:

```
-- Create Departments table

CREATE TABLE Departments (

    department_id INT PRIMARY KEY,

    department_name VARCHAR(100) NOT NULL

);



-- Alter Employees table to add foreign key constraint
```

ALTER TABLE Employees

ADD department_id INT,

ADD CONSTRAINT fk_department FOREIGN KEY (department_id) REFERENCES Departments(department_id);

7. Alter the `Employees` table to add a new column `phone_number` of type `VARCHAR(15)`.

Ans: ALTER TABLE Employees

ADD phone_number VARCHAR(15);

8. Drop the `phone_number` column from the `Employees` table.

Ans:

ALTER TABLE Employees

DROP COLUMN phone_number;

9. Drop the `Departments` table from the database.

Ans: DROP TABLE Departments;

10. Create a trigger `update_salary` that automatically updates the `salary` column in the `Employees` table to `6000` whenever a new employee is inserted with a salary less than `3000`.

Ans:

CREATE OR REPLACE TRIGGER update_salary

BEFORE INSERT ON Employees

FOR EACH ROW

BEGIN

    IF :NEW.salary < 3000 THEN

        :NEW.salary := 6000;

    END IF;

END;

11. Alter the `Employees` table to add a unique constraint on the `phone_number` column.

Ans: ALTER TABLE Employees

ADD CONSTRAINT unique_phone_number UNIQUE (phone_number);

12. Alter the `salary` column in the `Employees` table to increase its size to `DECIMAL(15, 2)`.

Ans:

ALTER TABLE Employees

MODIFY salary DECIMAL(15, 2);

13. Rename the `Employees` table to `Staff`.

Ans: rename employees to staff.

14. Create a view called `DepartmentSalarySummary` that shows the `department_id`, `department_name`, and the total `salary` of all employees in that department.

Ans:

CREATE VIEW DepartmentSalarySummary AS

SELECT d.department_id, d.department_name, SUM(e.salary) AS total_salary

FROM Employees e

JOIN Departments d ON e.department_id = d.department_id

GROUP BY d.department_id, d.department_name;

15. 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.

Ans:

CREATE INDEX idx_name_salary ON Employees (first_name, salary);

16. Create a sequence `course_seq` that starts at 1000, increments by 1, and has a minimum value of 1000 and a maximum value of 9999.

Ans:

```
CREATE SEQUENCE course_seq

START WITH 1000

INCREMENT BY 1

MINVALUE 1000

MAXVALUE 9999

CYCLE;
```

17. Create a table `Course_Enrollments` to track student course enrollments. The table should have:

- `enrollment_id` (primary key)
- `student_id` (foreign key referencing `Students.student_id`)
- `course_id` (foreign key referencing `Courses.course_id`)

Ans:

```
CREATE TABLE Course_Enrollments (

    enrollment_id INT PRIMARY KEY,

    student_id INT,

    course_id INT,

    enrollment_date DATE DEFAULT CURRENT_DATE,

    FOREIGN KEY (student_id) REFERENCES Students(student_id),

    FOREIGN KEY (course_id) REFERENCES Courses(course_id)

);
```

18. 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.

Ans:

```
ALTER TABLE Employees

ADD CONSTRAINT check_hire_date CHECK (hire_date <= CURRENT_DATE);
```

19. Drop the synonym `Emp` that was created for the `Employees` table.

Ans:

DROP SYNONYM Emp;

DML:

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`)

1. Get a list of all students in the university, showing their first and last names, email, and department name.
   Ans:
   SELECT s.first_name, s.last_name, s.email, d.department_name
   FROM Students s
   JOIN Departments d ON s.department_id = d.department_id;
2. Insert a new student into the `Students` table.
   Ans:
   INSERT INTO Students (student_id, first_name, last_name, email, department_id, enrollment_date, phone_number)
   VALUES (1002, 'Alice', 'Smith', 'alice.smith@example.com', 1, CURRENT_DATE, '555-1234');
3. Update the email address of the student with `student_id = 1002`.
   Ans:
   UPDATE Students
   SET email = 'alice.smith_new@example.com'
   WHERE student_id = 1002;
4. Delete a student record from the `Students` table where `student_id = 1002`.

   Ans:

   DELETE FROM Students

   WHERE student_id = 1002;

5. Retrieve all courses offered by the "Computer Science" department, including the course name and fee.
   Ans:
   SELECT c.course_name, c.course_fee
   FROM Courses c

JOIN Departments d ON c.department_id = d.department_id
WHERE d.department_name = 'Computer Science';

6. Find the names of all students enrolled in the course "Database Systems."

   Ans:

   SELECT s.first_name, s.last_name
   FROM Students s
   JOIN Enrollments e ON s.student_id = e.student_id
   JOIN Courses c ON e.course_id = c.course_id
   WHERE c.course_name = 'Database Systems';

7. 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'.

   Ans:

   INSERT INTO Enrollments (enrollment_id, student_id, course_id, enrollment_date, grade)
   VALUES (10001, 1001, 201, CURRENT_DATE, 'A');

8. Update the grade of student `student_id = 1001` for the course `course_id = 201` to 'B+'.

   Ans:

   UPDATE Enrollments
   SET grade = 'B+'
   WHERE student_id = 1001 AND course_id = 201;

9. Get the number of students enrolled in each course along with the course name.

   Ans:

   SELECT c.course_name, COUNT(e.student_id) AS num_students
   FROM Courses c
   LEFT JOIN Enrollments e ON c.course_id = e.course_id
   GROUP BY c.course_name;

10. Find all students who have not enrolled in any courses.

    Ans:

    SELECT s.first_name, s.last_name
    FROM Students s
    LEFT JOIN Enrollments e ON s.student_id = e.student_id
    WHERE e.enrollment_id IS NULL;

11. Retrieve the `first_name`, `last_name`, and `email` of all students enrolled in the "Mathematics" department.

Ans:

SELECT first_name, last_name, email

FROM Students

WHERE department_id = (SELECT department_id FROM Departments WHERE department_name = 'Mathematics');

12. Find all courses where the `course_fee` is greater than 2000, displaying the course name and fee.

Ans:

```
SELECT course_name, course_fee

FROM Courses

WHERE course_fee > 2000;
```

13. Retrieve a list of all students who have not yet paid their fees. Assume there is a `fee_paid` column in the `Students` table (boolean: `0` for not paid, `1` for paid).

Ans:

```
SELECT first_name, last_name, email

FROM Students

WHERE fee_paid = 0;
```

14. Count the number of courses each student is enrolled in, and display the student's name and the number of courses they are enrolled in.

Ans:

```
SELECT s.first_name, s.last_name, COUNT(e.enrollment_id) AS num_courses

FROM Students s

LEFT JOIN Enrollments e ON s.student_id = e.student_id

GROUP BY s.student_id;
```

15. Get the `first_name`, `last_name`, and `grade` of all students who are enrolled in the course "Advanced Databases."

    Ans:

    ```
    SELECT s.first_name, s.last_name, e.grade
    FROM Students s
    JOIN Enrollments e ON s.student_id = e.student_id
    JOIN Courses c ON e.course_id = c.course_id
    WHERE c.course_name = 'Advanced Databases';
    ```

16. Update the grade of all students enrolled in the course "Database Systems" to 'A+' if their current grade is 'B'.

    Ans:

    ```
    UPDATE Enrollments e
    SET grade = 'A+'
    WHERE e.course_id = (SELECT course_id FROM Courses WHERE course_name = 'Database
    Systems')
      AND e.grade = 'B';
    ```

17. Get a list of students who have completed the course "Data Structures" (i.e., those who have a grade other than NULL).

Ans:

SELECT s.first_name, s.last_name

FROM Students s

JOIN Enrollments e ON s.student_id = e.student_id

JOIN Courses c ON e.course_id = c.course_id

WHERE c.course_name = 'Data Structures' AND e.grade IS NOT NULL;

18. Find the department offering the course with the highest fee.

Ans:

SELECT d.department_name, c.course_name, c.course_fee

FROM Departments d

JOIN Courses c ON d.department_id = c.department_id

WHERE c.course_fee = (SELECT MAX(course_fee) FROM Courses);

19.

Insert multiple students into the `Students` table with their `student_id`, `first_name`, `last_name`, `email`, `department_id`, and `phone_number`.

Ans:

INSERT INTO Students (student_id, first_name, last_name, email, department_id, phone_number)

VALUES

(1003, 'Bob', 'Brown', 'bob.brown@example.com', 2, '555-5678'),

(1004, 'Charlie', 'Green', 'charlie.green@example.com', 3, '555-6789'),

(1005, 'Diana', 'White', 'diana.white@example.com', 1, '555-7890');

20.Delete students from the `Students` table who are not enrolled in any course (i.e., no entries in the `Enrollments` table).

Ans: DELETE FROM Students

WHERE student_id NOT IN (SELECT DISTINCT student_id FROM Enrollments);

MongoDB

1.Question:

Insert a document into the `products` collection with the following data:

- `name`: "Laptop"

- `category`: "Electronics"
- `price`: 1200
- `stock`: 50

Ans:

```
db.products.insertOne({
  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

```
db.orders.insertMany([
  {
    orderId: 101,
    customerName: "Alice",
    items: [{ name: "Laptop", quantity: 1 }, { name: "Mouse", quantity: 2 }],
    orderDate: new Date("2024-11-01")
  },
  {
    orderId: 102,
    customerName: "Bob",
    items: [{ name: "Phone", quantity: 1 }],
    orderDate: new Date("2024-11-02")
  }
]);
```

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

Ans: db.products.find({ category: "Electronics" }).toArray();

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

Ans:

db.products.find({ price: { $gt: 1000 } }).toArray();

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

Ans:

```
db.products.updateOne(
   { name: "Laptop" },  // Filter condition
   { $set: { stock: 45 } }  // Update operation
);
```

6. Update the stock of all products in the `Electronics` category by adding 10 to the existing stock.

Ans:

```
db.products.updateMany(
   { category: "Electronics" },  // Filter condition
   { $inc: { stock: 10 } }  // Increment the stock by 10
);
```

7. Delete the order with `orderId` 101 from the `orders` collection.

Ans:

db.orders.deleteOne({ orderId: 101 });

8. Remove all products from the `products` collection that belong to the `category` `"Electronics"`.

Ans: db.products.deleteMany({ category: "Electronics" });

9. 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" }

Ans: db.users.insertOne({

name: "Sara Lee",

email: "sara.lee@example.com",

age: 28,

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

});

10. 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

Ans:

db.posts.insertMany([

{

title: "Introduction to MongoDB",

author: "John",

content: "This is an introductory post about MongoDB.",

tags: ["MongoDB", "Database", "NoSQL"],

published: true

},

{

title: "Advanced MongoDB Queries",

author: "Jane",

content: "This post discusses advanced MongoDB query techniques.",

tags: ["MongoDB", "Queries", "Aggregation"],

published: false

}

]);

11. Query the `users` collection to find all users where `age` is greater than 30.

Ans:

```
db.users.find({ age: { $gt: 30 } }).toArray();
```

12. Query the `posts` collection to find all blog posts that are marked as `published: true`.

Ans:

```
db.posts.find({ published: true }).toArray();
```

13. Update the `address` of the user with `name: "Sara Lee"` to the following:

- `city`: "Los Angeles"
- `state`: "CA"

Ans:

```
db.users.updateOne(
  { name: "Sara Lee" },  // Filter condition
  { $set: { address: { city: "Los Angeles", state: "CA" } } }  // Update operation
);
```

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

Ans:

```
db.posts.updateOne(
  { title: "Introduction to MongoDB" },  // Filter condition
  { $addToSet: { tags: "Tutorial" } }  // Add the tag to the tags array
);
```

Q.Write a DDL query to create a table called `users` with the following columns:

- `user_id` (Primary Key, Integer, Auto-increment)
- `name` (Varchar, 100 characters)
- `email` (Varchar, 100 characters, Unique)
- `age` (Integer)

Ans: CREATE TABLE users (

user_id INT AUTO_INCREMENT PRIMARY KEY,

```
name VARCHAR(100),
email VARCHAR(100) UNIQUE,
age INT,
);
```