PostgreSQL Assignment

Assignment Description

In this assignment, you will work with PostgreSQL, a powerful open-source relational database management system. Your task involves creating 03 tables based on the provided sample data and then writing and executing queries to perform various database operations such as creating, reading, updating, and deleting data. Additionally, you will explore concepts like LIMIT and OFFSET, JOIN operations, GROUP BY, aggregation and LIKE.

Instructions:

Database Setup:

- Create a fresh database titled **"university_db"** or any other appropriate name.

Table Creation:

Create a **"students"** table with the following fields:

- student_id (Primary Key): Integer, unique identifier for students.
- student_name: String, representing the student's name.
- age: Integer, indicating the student's age.
- email: String, storing the student's email address.
- frontend_mark: Integer, indicating the student's frontend assignment marks.
- backend_mark: Integer, indicating the student's backend assignment marks.
- status: String, storing the student's result status.

```
postgres=# create database nsw_university;
CREATE DATABASE
postgres=# \c nsw_university
You are now connected to database "nsw_university" as user "postgres".
nsw_university=# CREATE TABLE students (
nsw_university(#     student_id INT PRIMARY KEY,
nsw_university(#     student_name VARCHAR(100),
nsw_university(#     age INT,
nsw_university(#     email VARCHAR(100),
nsw_university(#     frontend_mark INT,
nsw_university(#     backend_mark INT,
nsw_university(#     status VARCHAR(50)
nsw_university(# );
CREATE TABLE
```

Create a **"courses"** table with the following fields:

- course_id (Primary Key): Integer, unique identifier for courses.
- course_name: String, indicating the course's name.

- credits: Integer, signifying the number of credits for the course.

```
nsw_university=# CREATE TABLE courses (
nsw_university(#     course_id INT PRIMARY KEY,
nsw_university(#     course_name VARCHAR(100),
nsw_university(#     credits INT
nsw_university(# );
CREATE TABLE
```

Create an **"enrollment"** table with the following fields:

- enrollment_id (Primary Key): Integer, unique identifier for enrollments.
- student_id (Foreign Key): Integer, referencing student_id in "Students" table.
- course_id (Foreign Key): Integer, referencing course_id in "Courses" table.

```
nsw_university=# CREATE TABLE enrollment (
nsw_university(#     enrollment_id INT PRIMARY KEY,
nsw_university(#     student_id INT,
nsw_university(#     course_id INT,
nsw_university(#     FOREIGN KEY (student_id) REFERENCES students(student_id),
nsw_university(#     FOREIGN KEY (course_id) REFERENCES courses(course_id)
nsw_university(# );
CREATE TABLE
```

Sample Data

- Insert the following sample data into the **"students"** table:

student_id	student_name	age	email	frontend_mark	backend_mark	status
1	Alice	22	alice@example.com	55	57	NULL
2	Bob	21	bob@example.com	34	45	NULL
3	Charlie	23	charlie@example.com	60	59	NULL
4	David	20	david@example.com	40	49	NULL

student_id	student_name	age	email	frontend_mark	backend_mark	status
5	Eve	24	newemail@example.com	45	34	NULL
6	Rahim	23	rahim@gmail.com	46	42	NULL

```
nsw_university=# INSERT INTO students (student_id, student_name, age, email, frontend_mark, backend_mark, status)
nsw_university-# VALUES
nsw_university-#     (1, 'Suresh', 24, 'suresh@gmail.com', 54, 57, NULL),
nsw_university-#     (2, 'Dhanesh', 22, 'dhanesh@gmail.com', 44, 45, NULL),
nsw_university-#     (3, 'Deva', 23, 'deva@gmail.com', 60, 69, NULL),
nsw_university-#     (4, 'Manoj', 24, 'manjo@gmail.com', 47, 49, NULL),
nsw_university-#     (5, 'Ravi', 24, 'ravi@gmail.com', 85, 37, NULL),
nsw_university-#     (6, 'Rahul', 21, 'rahul@gmail.com', 47, 48, NULL);
INSERT 0 6
```

- Insert the following sample data into the **"courses"** table:

course_id	course_name	credits
1	Next.js	3
2	React.js	4
3	Databases	3
4	Prisma	3

```
nsw_university=# INSERT INTO courses (course_id, course_name, credits)
nsw_university-# VALUES
nsw_university-#     (1, 'Next.js', 3),
nsw_university-#     (2, 'React.js', 4),
nsw_university-#     (3, 'Databases', 3),
nsw_university-#     (4, 'Prisma', 3);
INSERT 0 4
```

- Insert the following sample data into the **"enrollment"** table:

enrollment_id	student_id	course_id
1	1	1
2	1	2
3	2	1
4	3	2

```
nsw_university=# INSERT INTO enrollment (enrollment_id, student_id, course_id)
nsw_university-# VALUES
nsw_university-#     (1, 1, 1),
nsw_university-#     (2, 1, 2),
nsw_university-#     (3, 2, 1),
nsw_university-#     (4, 3, 2);
INSERT 0 4
```

Execute SQL queries to fulfill the ensuing tasks:

Query 1:

Insert a new student record with the following details:

- Name: YourName
- Age: YourAge
- Email: YourEmail
- Frontend-Mark: YourMark
- Backend-Mark: YourMark
- Status: NULL

```
nsw_university=# INSERT INTO students (student_name, age, email, frontend_mark, backend_mark, status)
nsw_university-# VALUES ('sureshmanikandan', 23, 'sureshmk@gmail.com', 99, 91, NULL);
ERROR:  null value in column "student_id" of relation "students" violates not-null constraint
DETAIL:  Failing row contains (null, sureshmanikandan, 23, sureshmk@gmail.com, 99, 91, null).
nsw_university=# CREATE SEQUENCE student_id START WITH 7 INCREMENT BY 1;
CREATE SEQUENCE
nsw_university=# ALTER TABLE students ALTER COLUMN student_id SET DEFAULT nextval('student_id');
ALTER TABLE
nsw_university=# INSERT INTO students (student_name, age, email, frontend_mark, backend_mark, status)
nsw_university-# VALUES ('sureshmanikandan', 23, 'sureshmk@gmail.com', 99, 91, NULL);
INSERT 0 1
nsw_university=# select*from students;
 student_id |   student_name    | age |       email        | frontend_mark | backend_mark | status
------------+-------------------+-----+--------------------+---------------+--------------+--------
          1 | Suresh            |  24 | suresh@gmail.com   |            54 |           57 |
          2 | Dhanesh           |  22 | dhanesh@gmail.com  |            44 |           45 |
          3 | Deva              |  23 | deva@gmail.com     |            60 |           69 |
          4 | Manoj             |  24 | manjo@gmail.com    |            47 |           49 |
          5 | Ravi              |  24 | ravi@gmail.com     |            85 |           37 |
          6 | Rahul             |  21 | rahul@gmail.com    |            47 |           48 |
          7 | sureshmanikandan  |  23 | sureshmk@gmail.com |            99 |           91 |
(7 rows)
```

Query 2:

Retrieve the names of all students who are enrolled in the course titled 'Next.js'.

Sample Output:

student_name

Alice

Bob

```
nsw_university=# SELECT s.student_name
nsw_university-# FROM students s
nsw_university-# JOIN enrollment e ON s.student_id = e.student_id
nsw_university-# JOIN courses c ON e.course_id = c.course_id
nsw_university-# WHERE c.course_name = 'Next.js';
 student_name
--------------
 Suresh
 Dhanesh
(2 rows)
```

Query 3:

Update the status of the student with the highest total (frontend_mark + backend_mark) mark to 'Awarded'

```
nsw_university=# UPDATE students
nsw_university-# SET status = 'Cash Awarded'
nsw_university-# WHERE (frontend_mark + backend_mark) = (
nsw_university(#     SELECT MAX(frontend_mark + backend_mark) FROM students
nsw_university(# );
UPDATE 1
nsw_university=# select*from students;
 student_id |   student_name    | age |       email        | frontend_mark | backend_mark |    status
------------+-------------------+-----+--------------------+---------------+--------------+--------------
          1 | Suresh            |  24 | suresh@gmail.com   |            54 |           57 |
          2 | Dhanesh           |  22 | dhanesh@gmail.com  |            44 |           45 |
          3 | Deva              |  23 | deva@gmail.com     |            60 |           69 |
          4 | Manoj             |  24 | manjo@gmail.com    |            47 |           49 |
          5 | Ravi              |  24 | ravi@gmail.com     |            85 |           37 |
          6 | Rahul             |  21 | rahul@gmail.com    |            47 |           48 |
          7 | sureshmanikandan  |  23 | sureshmk@gmail.com |            99 |           91 | Cash Awarded
(7 rows)
```

Query 4:

Delete all courses that have no students enrolled.

```
nsw_university=# DELETE FROM courses
nsw_university-# WHERE course_id NOT IN (
nsw_university(#     SELECT DISTINCT course_id FROM enrollment
nsw_university(# );
DELETE 2
nsw_university=# select*from courses;
 course_id | course_name | credits
-----------+-------------+---------
         1 | Next.js     |       3
         2 | React.js    |       4
(2 rows)
```

Query 5:

Retrieve the names of students using a limit of 2, starting from the 3rd student.

Sample Output:

student_name

Charlie

David

```
nsw_university=# SELECT student_name
nsw_university-# FROM students
nsw_university-# ORDER BY student_id
nsw_university-# LIMIT 2 OFFSET 2;
 student_name
--------------
 Deva
 Manoj
(2 rows)
```

Query 6:

Retrieve the course names and the number of students enrolled in each course.

Sample Output:

course_name	students_enrolled
Next.js	2
React.js	2

```
nsw_university=# SELECT c.course_name, COUNT(e.student_id) AS students_enrolled
nsw_university-# FROM courses c
nsw_university-# LEFT JOIN enrollment e ON c.course_id = e.course_id
nsw_university-# GROUP BY c.course_name;
 course_name | students_enrolled
-------------+-------------------
 Next.js     |                 2
 React.js    |                 2
(2 rows)
```

Query 7:

Calculate and display the average age of all students.

Sample Output:

average_age
22.2857142857142857

```
nsw_university=# SELECT AVG(age) AS average_age
nsw_university-# FROM students;
     average_age
---------------------
 23.0000000000000000
(1 row)
```

Query 8:

Retrieve the names of students whose email addresses contain 'example.com'.

Sample Output:

student_name

Alice

Bob

Charlie

`David

```
nsw_university=# SELECT student_name
nsw_university-# FROM students
nsw_university-# WHERE email LIKE '%gmail.com%';
   student_name
------------------
 Suresh
 Dhanesh
 Deva
 Manoj
 Ravi
 Rahul
 sureshmanikandan
(7 rows)
```

Prepare the SQL code for table creation, sample data insertion, and the seven queries in a text document or your preferred format. Include comments explaining each query's purpose and functionality. **Save your document as "PostgreSQL_Assignment.sql" or any other appropriate name.**

Based on the above table data explain the concept along with the example for below items

1. Explain the primary key and foreign key concepts in PostgreSQL.
   Primary Key:
   A primary key is a uniquely identifies each row in a table.
   It cannot have NULL values and must be unique for each row.
   Foreign Key:
   A foreign key is a column that creates a relationship between two tables.
   It references the primary key in another table to ensure data consistency.

2. What is the difference between the VARCHAR and CHAR data types?
   VARCHAR: Variable-length string. Uses only as much space as needed.
   CHAR: Fixed-length string. Always uses the specified amount of space, padding with spaces if necessary.

3. Explain the purpose of the WHERE clause in a SELECT statement.
   WHERE keyword is used for conditional statements on a table or in between tables for filter records.

4. What are the LIMIT and OFFSET clauses used for?
   LIMIT is used to restricts the number of rows on a query.
   OFFSET is used to skips number of rows before start.

5. How can you perform data modification using UPDATE statements?
   The UPDATE statement changes existing records in a table.

   ```
   UPDATE students
   SET student_name = sureshkumar
   WHERE student_id = 1;
   ```

6. What is the significance of the JOIN operation, and how does it work in PostgreSQL?
   JOIN combines rows from two or more tables based on a similar column between those tables.

```
SELECT s.student_name, c.course_name
FROM students
JOIN enrollment ON s.student_id = e.student_id
JOIN courses ON e.course_id = c.course_id;
```

7. Explain the GROUP BY clause and its role in aggregation operations.
   GROUP BY that have the same values in specified columns into summary rows.
   Aggregate functions are COUNT, SUM, AVG, MIN,MAX.

8. How can you calculate aggregate functions like COUNT, SUM, and AVG in PostgreSQL?
   SELECT COUNT(*) FROM students;
   SELECT SUM(backend_mark) FROM students;
   SELECT AVG(age) FROM students;

9. What is the purpose of an index in PostgreSQL, and how does it optimize query performance?
   An index that helps to improves the speed of data retrieval from a database table.

   It works like an index in a book, allowing quick look-ups when using query for the table to retrive the data from the table.

10. Explain the concept of a PostgreSQL view and how it differs from a table.

    A view is a virtual table on the result of a SELECT query.

    A view does not store data itself but displays data stored in other tables.