```
 CREATE DATABASE university_db
postgres-# ;
CREATE DATABASE
postgres=# \c university_db
You are now connected to database "university_db" as user "postgres".
university_db=#
```

```
university_db=# CREATE TABLE students (
university_db(#     student_id SERIAL PRIMARY KEY,  // Using SERIAL to generate
the student_id automatically in a sequence
university_db(#     student_name VARCHAR(100),
university_db(#     age INTEGER,
university_db(#     email VARCHAR(100),
university_db(#     frontend_mark INTEGER,
university_db(#     backend_mark INTEGER,
university_db(#     status VARCHAR(50));
CREATE TABLE
```

```
university_db=# CREATE TABLE courses (
university_db(#     course_id SERIAL PRIMARY KEY,
university_db(#     course_name VARCHAR(100),
university_db(#     credits INTEGER
university_db(# );
CREATE TABLE
```

```
university_db=# CREATE TABLE enrollment (
university_db(#     enrollment_id SERIAL PRIMARY KEY,
university_db(#     student_id INTEGER REFERENCES students(student_id),
university_db(#     course_id INTEGER REFERENCES courses(course_id)
university_db(# );
```

CREATE TABLE

```
university_db=# INSERT INTO students (student_name, age, email, frontend_mark,
backend_mark, status)
university_db-#   VALUES
university_db-#      ('Senid', 24, 'senid@gmail.com', 75, 85, NULL),
university_db-#      ('Vikram', 25, 'vikram@gmail.com', 66, 54, NULL),
university_db-#      ('Denver', 28, 'denver@gmail.com', 64, 76, NULL),
university_db-#      ('tokyo', 27, 'tokyo@gmail.com', 76, 82, NULL),
university_db-#      ('oslo', 29, 'oslo@gmail.com', 72, 74, NULL),
university_db-#      ('nirobi', 32, 'nirobi@gmail.com', 67, 81, NULL);
INSERT 0 6


university_db=# INSERT INTO courses (course_name, credits)
university_db-#   VALUES
university_db-#      ('Next.js', 3),
university_db-#      ('React.js', 4),
university_db-#      ('Databases', 3),
university_db-#      ('Prisma', 3);
INSERT 0 4


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

==============================QUERY1============================

1)

```
university_db=# INSERT INTO students (student_name, age, email, frontend_mark,
backend_mark, status)
university_db-#   VALUES ('palanisamy', 35, 'palanisamy@gmail.com', 99, 98, NULL);
INSERT 0 1
```

===============================QUERY2============================

2)

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

===============================QUERY3============================

3)

```
university_db=# UPDATE students
university_db-#   SET status = 'Awarded'
university_db-#   WHERE student_id = (
university_db(#     SELECT student_id
university_db(#     FROM (
university_db(#         SELECT student_id, (frontend_mark + backend_mark) AS total_mark
university_db(#         FROM students
university_db(#         ORDER BY total_mark DESC
university_db(#         LIMIT 1
university_db(#     ) AS highest_mark
university_db(#  );
UPDATE 1
```

===============================QUERY4============================

4)

```
university_db=# DELETE FROM courses
university_db-#   WHERE course_id NOT IN (SELECT DISTINCT course_id FROM
enrollment);
DELETE 2
```

===============================QUERY5============================

5)

```
university_db=# SELECT student_name
university_db-#   FROM students
university_db-#   ORDER BY student_id
university_db-#   LIMIT 2 OFFSET 2;
 student_name
--------------
 Denver
 tokyo
(2 rows)
```

===============================QUERY6============================

6)

```
university_db=# SELECT c.course_name, COUNT(e.student_id) AS
students_enrolled
university_db-#   FROM courses c
university_db-#   LEFT JOIN enrollment e ON c.course_id = e.course_id
university_db-#   GROUP BY c.course_name;
 course_name  | students_enrolled
```

```
-------------+-------------------
 Next.js     |                 2
 React.js    |                 2
(2 rows)
```

============================QUERY7=============================
7)

```
university_db=# SELECT AVG(age) AS average_age
university_db-#   FROM students;
     average_age
---------------------
 28.5714285714285714
(1 row)
```

==============================QUERY8===========================
8)

```
university_db=# SELECT student_name
university_db-#   FROM students
university_db-#   WHERE email LIKE '%gmail.com';
 student_name
--------------
 Senid
 Vikram
 Denver
 tokyo
 oslo
 nirobi
 palanisamy
(7 rows)
```

===

1. Explain the primary key and foreign key concepts in PostgreSQL.

PRIMARY KEY : a primary key uniquely identifies each record in a table. It must contain unique values and cannot be null.
EXAMPLE : In the students table, student_id serves as the primary key because each student has a unique ID.

```
university_db-# ;
 student_id
------------
          1
          2
          3
          4
          5
          6
          7
(7 rows)
```

Foreign Key: a foreign key represents a relationship between two tables. It refers to the primary key of another table.
EXAMPLE :In the enrollment table,student_id and course_id are foreign keys referencing the Students and courses tables respectively.

****

2. What is the difference between the VARCHAR and CHAR data types?

VARCHAR: Stores variable-length character strings. It can hold up to a specified length but only uses as much storage as needed. varchar data type contain all characters

EXAMPLE : email in the students table is a varchar type.

CHAR: Stores fixed-length character strings char data type also contain all characters

EXAMPLE : a CHAR(10) column would always occupy 10 characters

****

3. Explain the purpose of the WHERE clause in a SELECT statement.

The where clause filters records based on a specified condition. It allows you to retrieve only the rows that meet certain criteria.
EXAMPLE :

```
university_db=# SELECT * FROM students WHERE age > 20;
 student_id | student_name | age |        email         | frontend_mark | backend_mark | status
------------+--------------+-----+----------------------+---------------+--------------+---------
          1 | Senid        |  24 | senid@gmail.com      |            75 |           85 |
          2 | Vikram       |  25 | vikram@gmail.com     |            66 |           54 |
          3 | Denver       |  28 | denver@gmail.com     |            64 |           76 |
          4 | tokyo        |  27 | tokyo@gmail.com      |            76 |           82 |
          5 | oslo         |  29 | oslo@gmail.com       |            72 |           74 |
          6 | nirobi       |  32 | nirobi@gmail.com     |            67 |           81 |
          7 | palanisamy   |  35 | palanisamy@gmail.com |            99 |           98 | Awarded
(7 rows)
```

****

4. What are the LIMIT and OFFSET clauses used for?

LIMIT: Limits the number of rows returned in a query result. Useful for pagination or restricting large result sets.

OFFSET: Specifies where to start returning rows from. Combined with Limit .it enables pagination by skipping a certain number of rows.

EXAMPLE :

```
university_db=# SELECT * FROM students LIMIT 3 OFFSET 2;
 student_id | student_name | age |      email       | frontend_mark | backend_mark |
status
------------+--------------+-----+------------------+---------------+--------------+--------
          3 | Denver       |  28 | denver@gmail.com |            64 |           76 |
          4 | tokyo        |  27 | tokyo@gmail.com  |            76 |           82 |
          5 | oslo         |  29 | oslo@gmail.com   |            72 |           74 |
(3 rows)
```

****

5. How can you perform data modification using UPDATE statements?

UPDATE : statements modify existing records in a table. They change the values of specific columns based on a condition.

EXAMPLE : UPDATE students SET age = 30 WHERE student_id = 2;
****

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 related column between them. It allows you to retrieve data from multiple tables in a single query.

EXAMPLE :

```
university_db=# SELECT students.student_name, courses.course_name
university_db-# FROM students
university_db-# JOIN enrollment ON students.student_id = enrollment.student_id
university_db-# JOIN courses ON enrollment.course_id = courses.course_id;
 student_name | course_name
--------------+-------------
```

```
 Senid        | Next.js
 Senid        | React.js
 Vikram       | Next.js
 Denver       | React.js
(4 rows)
```

****

7 .Explain the GROUP BY clause and its role in aggregation operations.

GROUP BY:

Groups rows that have the same values into summary rows. It's used with aggregate functions like COUNT ,SUM,AVG etc., to perform calculations on grouped data.

EXAMPLE :

```
university_db=# SELECT status, COUNT(*) AS count_students

university_db-# FROM students

university_db-# GROUP BY status;

 status   | count_students

---------+----------------

         |            6

 Awarded |            1

(2 rows)
```

****

8. How can you calculate aggregate functions like COUNT, SUM, and AVG in PostgreSQL?

COUNT: Counts the number of rows returned by a query.
SUM: Calculates the sum of numeric values in a column.
AVG: Computes the average of numeric values in a column.

EXAMPLE :

```
university_db=# SELECT COUNT(*) AS total_students, SUM(age) AS total_age, AVG(age) AS avg_age
university_db-# FROM students;
 total_students | total_age |       avg_age
----------------+-----------+---------------------
              7 |       205 | 29.2857142857142857
(1 row)
```

****

9. What is the purpose of an index in PostgreSQL, and how does it optimize query performance?

Index: Improves the speed of data retrieval operations on a database table by providing quick access to rows based on the indexed column(s).

EXAMPLE :

```
CREATE INDEX idx_student_name ON students(student_name);
```

****

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

View: A virtual table generated from a SELECT query. It represents a subset of data from one or more tables, similar to a saved query. Views do not store data themselves; they display data from underlying tables.

Table: Stores actual data. Tables are the fundamental storage unit in a relational database.

EXAMPLE :

```
CREATE VIEW student_courses AS
SELECT students.student_name, courses.course_name
FROM students
JOIN enrollment ON students.student_id = enrollment.student_id
JOIN courses ON enrollment.course_id = courses.course_id;
```

****