PostgreAssignment 1 ASSIGNMENT

CREATE DATABASE university\_db;

\C university\_db

======================================================================================================================

CREATE TABLE students (

student\_id SERIAL PRIMARY KEY,

student\_name VARCHAR(80),

age INTEGER,

email VARCHAR(80),

frontend\_mark INTEGER,

backend\_mark INTEGER,

status VARCHAR(50)

);

=====================================================================================================================

CREATE TABLE courses(

course\_id SERIAL PRIMARY KEY,

course\_name VARCHAR(100),

credits INTEGER

);

=======================================================================================================================

CREATE TABLE enrollment (

enrollment\_id SERIAL PRIMARY KEY,

student\_id INTEGER REFERENCES students(student\_id),

course\_id INTEGER REFERENCES courses(course\_id)

);

==========================================================================================================================

INSERT INTO students (student\_name, age, email, frontend\_mark, backend\_mark, status)

VALUES

('Alice', 22, 'alice@example.com', 55, 57, NULL),

('Bob', 21, 'bob@example.com', 34, 45, NULL),

('Charlie', 23, 'charlie@example.com', 60, 59, NULL),

('David', 20, 'david@example.com', 40, 49, NULL),

('Eve', 24, 'newemail@example.com', 45, 34, NULL),

('Rahim', 23, 'rahim@gmail.com', 46, 42, NULL);

Ans

university\_db=# select \* from students;

student\_id | student\_name | age | email | frontend\_mark | backend\_mark | status

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

1 | Alice | 22 | alice@example.com | 55 | 57 |

2 | Bob | 21 | bob@example.com | 34 | 45 |

3 | Charlie | 23 | charlie@example.com | 60 | 59 |

4 | David | 20 | david@example.com | 40 | 49 |

5 | Eve | 24 | newemail@example.com | 45 | 34 |

6 | Rahim | 23 | rahim@gmail.com | 46 | 42 |

(6 rows)

============================================================================================================================

INSERT INTO courses (course\_name, credits)

VALUES

('Next.js', 3),

('React.js', 4),

('Databases', 3),

('Prisma', 3);

INSERT 0 4

=========================================================================================================================

INSERT INTO enrollment (student\_id, course\_id)

VALUES

(1, 1),

(1, 2),

(2, 1),

(3, 2);

university\_db=# select \* from enrollment;

enrollment\_id | student\_id | course\_id

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

1 | 1 | 1

2 | 1 | 2

3 | 2 | 1

4 | 3 | 2

(4 rows)

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

ANS:

INSERT INTO students (student\_name, age, email, frontend\_mark, backend\_mark, status)

VALUES ('DHARANESH', 23, 'DHARAN@GMAIL.COM', 88, 90, NULL);

university\_db=# INSERT INTO students (student\_name, age, email, frontend\_mark, backend\_mark, status)

university\_db-# VALUES ('DHARANESH', 23, 'DHARAN@GMAIL.COM', 88, 90, NULL);

INSERT 0 1

university\_db=# select \* from students;

student\_id | student\_name | age | email | frontend\_mark | backend\_mark | status

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

1 | Alice | 22 | alice@example.com | 55 | 57 |

2 | Bob | 21 | bob@example.com | 34 | 45 |

3 | Charlie | 23 | charlie@example.com | 60 | 59 |

4 | David | 20 | david@example.com | 40 | 49 |

5 | Eve | 24 | newemail@example.com | 45 | 34 |

6 | Rahim | 23 | rahim@gmail.com | 46 | 42 |

7 | DHARANESH | 23 | DHARAN@GMAIL.COM | 88 | 90 | Awarded

(7 rows)

==========================================================================================================================

Query 2:

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

ANS

SELECT s.student\_name

FROM students s

JOIN enrollment e ON s.student\_id = e.student\_id

JOIN courses c ON e.course\_id = c.course\_id

WHERE c.course\_name = 'Next.js';

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

--------------

Alice

Bob

(2 rows)

=============================================================================================================================

Query 3:

Update the status of the student with the highest total (frontend\_mark + backend\_mark) mark to 'Awarded'

ANS

UPDATE students

SET status = 'Awarded'

WHERE student\_id = (

SELECT student\_id

FROM (

SELECT student\_id, (frontend\_mark + backend\_mark) AS total\_mark

FROM students

ORDER BY total\_mark DESC

LIMIT 1

) AS highest\_mark);

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);

UPDATE 1

=====================================================================================================================

Query 4:

Delete all courses that have no students enrolled.

ANS

DELETE FROM courses

WHERE course\_id NOT IN (SELECT DISTINCT course\_id FROM enrollment);

university\_db=# DELETE FROM courses

university\_db-# WHERE course\_id NOT IN (SELECT DISTINCT course\_id FROM enrollment);

DELETE 2

========================================================================================================================

Query 5:

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

Sample Output:

student\_name

Charlie

David

ANS

SELECT student\_name

FROM students

ORDER BY student\_id

LIMIT 2 OFFSET 2;

university\_db=# SELECT student\_name

university\_db-# FROM students

university\_db-# ORDER BY student\_id

university\_db-# LIMIT 2 OFFSET 2;

student\_name

--------------

Charlie

David

(2 rows)

=========================================================================================================================

Query 6:

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

ANS

SELECT c.course\_name, COUNT(e.student\_id) AS students\_enrolled

FROM courses c

LEFT JOIN enrollment e ON c.course\_id = e.course\_id

GROUP BY c.course\_name;

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

Next.js | 2

React.js | 2

(2 rows)

============================================================================================================

Query 7:

Calculate and display the average age of all students.

SELECT AVG(age) AS average\_age

FROM students;

university\_db=# SELECT AVG(age) AS average\_age

university\_db-# FROM students;

average\_age

---------------------

22.2857142857142857

(1 row)

======================================================================================================================

Query 8:

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

SELECT student\_name

FROM students

WHERE email LIKE '%example.com';

university\_db=# SELECT student\_name

university\_db-# FROM students

university\_db-# WHERE email LIKE '%example.com';

student\_name

--------------

Alice

Bob

Charlie

David

Eve

(5 rows)

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

Primary key is used to find the uniquely from the table and foreign key is used to established the connection between the two tables Primary key used in column to increment the values antomatically and forigen key is used update value in column exists in the indexed column in different tables.

Example **is** in the students table, student\_id is the primary key because it uniquely identifies each student. The enrollment table, student\_id and course\_id are foreign keys that reference student\_id in students table and course\_id in courses table respectively

==========================================================================

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

Varchar datatype contain all characters and the variable length allocation take place.varaible -length of column data

Char data type also contain all characters and this type is fixed in length

\*VARCHAR(100) in the students table allows storing email addresses up to 100 characters long.

\*status CHAR(50) in the students table will store status strings exactly 50 characters long, padding shorter strings with spaces.

===============================================================================

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

Where:where clause is used to filter the records and it was execute row by row.

select:select clause is used to display the records according to the conduction.

\*WHERE email LIKE '%example.com' filters students whose email addresses end with 'example.com' in the students table.

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

Limiit: limit is used to give the limited rows according to number you enter in query

Offset:offset is used for which row you want to start the record data

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

Syntax:

update table\_name

set coloumn\_name = “values”

(or)

update table\_name

set coloumn\_name = “values”

where conduction;

\*UPDATE students SET status = 'Awarded' WHERE ... updates the status of students who meet certain criteria (e.g., highest marks).

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 related columns. Allows retrieval of related data from multiple tables in a single query.

LEFT JOIN enrollment ON c.course\_id = e.course\_id retrieves courses with the count of students enrolled.

\*join enrollment e ON s.student\_id = e.student\_id links students and enrollment tables based on student\_id.

==================================================================================

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

group by clause is used to group the records according to the column name and aggregation function for the example we can use the avg() function to aggregation function and perform special operations on entire table or on a set,or group of rows rather than on each row and then return one row of values for each group.

\*GROUP BY course\_name in SELECT course\_name, COUNT(\*) AS students\_enrolled calculates the number of students enrolled per course.

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 values in a column.

AVG: Calculates the average of values in a column.

Example: SELECT AVG(age) AS average\_age FROM students calculates the average age of all students.

================================================================================

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

Index: Improves query performance by reducing the number of data pages PostgreSQL needs to read. Speeds up data retrieval operations.

Example: CREATE INDEX idx\_student\_email ON students(email); creates an index on the email column for faster email-based searches

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

By using the views we can reduce the code size and code efficiency