POSTGRESQL-ASSIGNTMENT:

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

# Create new database

CREATE DATABASE **UNIVERSITY\_DB;**

//Droping the table if the table is already exits.

DROP TABLE IF EXISTS STUDENT;

# //CREATE TABLE STUDENT

# 

CREATE TABLE STUDENT(

STUDENT\_ID SERIAL NOT NULL UNIQUE PRIMARY KEY,

STUDENT\_NAME VARCHAR(100) NOT NULL UNIQUE ,

AGE INT NOT NULL,

EMAIL VARCHAR(20) NOT NULL UNIQUE,

FRONTEND\_MARK INT NOT NULL,

BACKEND\_MARK INT NOT NULL,

STATUS VARCHAR(50) );

## // INSERTING AN VALUES INTO THE TABLE STUDENT.

INSERT INTO STUDENT ( STUDENT\_NAME, AGE, EMAIL, FRONTEND\_MARK, BACKEND\_MARK, STATUS)

VALUES

( 'SIVA', 22, 'siva@example.com', 88, 95, 'Pass'),

('BASIL', 21, 'basil@example.com', 78, 80, 'Pass'),

( 'SANDEEP', 21, 'sandeep@example.com', 90, 81, 'Pass'),

( 'DINESH', 21, 'dinesh@example.com', 72, 82, 'Pass'),

( 'EASHWAR', 21, 'eashwar@example.com', 68, 60, 'Pass'),

( 'RAM', 21, 'ram@example.com', 68, 70, 'Pass'),

( 'RAVI', 23, 'ravi@example.com', 92, 95, 'Pass');

# // SELECT \* FROM STUDENT;

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

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

1 | SIVA | 22 | siva@example.com | 88 | 95 | Pass

2 | BASIL | 21 | basil@example.com | 78 | 80 | Pass

3 | SANDEEP | 21 | sandeep@example.com | 90 | 81 | Pass

4 | DINESH | 21 | dinesh@example.com | 72 | 82 | Pass

5 | EASHWAR | 21 | eashwar@example.com | 68 | 60 | Pass

6 | RAM | 21 | ram@example.com | 68 | 70 | Pass

7 | RAVI | 23 | ravi@example.com | 92 | 95 | Pass

(7 rows)

//CREATE TABLE COURSES

CREATE TABLE COURSES (

COURSES\_ID SERIAL NOT NULL UNIQUE PRIMARY KEY,

COURSE\_NAME VARCHAR(50) NOT NULL UNIQUE ,

CREDITS INT NOT NULL );

## // INSERTING AN VALUES INTO THE TABLE COURSES.

INSERT INTO COURSES ( COURSE\_NAME, CREDITS)

VALUES

( 'PYTHON',4),

( 'JAVA',5),

( 'REACT',3),

( '.NET',2),

( 'C++',4);

// SELECT \* FROM COURSES;

**courses\_id | course\_name | credits**

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

1 | PYTHON | 4

2 | JAVA | 5

3 | REACT | 3

4 | .NET | 2

5 | C++ | 4

(5 rows)

//CREATE TABLE ENROLLMENT

CREATE TABLE ENROLLMENT (

ENROLLMENT\_ID SERIAL NOT NULL UNIQUE PRIMARY KEY,

STUDENT\_ID INT,

COURSE\_ID INT,

FOREIGN KEY (STUDENT\_ID) REFERENCES STUDENT(STUDENT\_ID),

FOREIGN KEY (COURSE\_ID) REFERENCES COURSES(COURSES\_ID)

);

## // INSERTING AN VALUES INTO THE TABLE ENROLLMENT.

INSERT INTO ENROLLMENT ( STUDENT\_ID, COURSE\_ID)

VALUES

( 1,4),

( 1,5),

( 3,3),

( 2,2),

( 4,4);

// SELECT \* FROM ENROLLMENT;

**enrollment\_id | student\_id | course\_id**

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

1 | 1 | 4

2 | 1 | 5

3 | 3 | 3

4 | 2 | 2

5 | 4 | 4

(5 rows)

# 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**

**)**

INSERT INTO STUDENT ( STUDENT\_NAME, AGE, EMAIL, FRONTEND\_MARK, BACKEND\_MARK, STATUS)

VALUES

( 'VICTOR', 23, 'victor@example.com', 100, 100, NULL);

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

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

8 | VICTOR | 23 | victor@example.com | 100 | 100 |

# QUERY 2:

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

**Retrieve the names of all students who are enrolled in the course titled 'JAVA'.**

SELECT STUDENT\_NAME

FROM STUDENT

WHERE STUDENT\_ID IN (

SELECT STUDENT\_ID

FROM ENROLLMENT

WHERE COURSE\_ID = (

SELECT COURSES\_ID

FROM COURSES

WHERE COURSE\_NAME = 'JAVA'

)

);

**student\_name**

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

BASIL

(1 row)

# QUERY 3:

**============**

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

UPDATE STUDENT

SET STATUS = 'AWARDED'

WHERE STUDENT\_ID = (

SELECT STUDENT\_ID

FROM STUDENT

ORDER BY (FRONTEND\_MARK + BACKEND\_MARK) DESC

LIMIT 1

);

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

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

8 | VICTOR | 23 | victor@example.com | 100 | 100 | AWARDED

(1 row)

# QUERY 4:

-=======

**Delete all courses that have no students enrolled.**

DELETE FROM COURSES

WHERE COURSES\_ID NOT IN (

SELECT DISTINCT COURSES\_ID

FROM ENROLLMENT

);

**courses\_id | course\_name | credits**

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

1 | PYTHON | 4

2 | JAVA | 5

3 | REACT | 3

4 | .NET | 2

5 | C++ | 4

(5 rows)

# QUERY 5:

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

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

SELECT STUDENT\_NAME FROM STUDENT

OFFSET 2

LIMIT 2 ;

student\_name

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

SANDEEP

DINESH

(2 rows)

# QUERY 6:

===========

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

SELECT C.COURSE\_NAME, COUNT(E.STUDENT\_ID) AS STUDENTS\_ENROLLED

FROM COURSES C

LEFT JOIN ENROLLMENT E

ON

C.COURSES\_ID = E.COURSE\_ID

GROUP BY C.COURSE\_NAME

ORDER BY C.COURSE\_NAME;

**course\_name | students\_enrolled**

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

.NET | 2

C++ | 1

JAVA | 1

PYTHON | 0

REACT | 1

(5 rows)

# QUERY 7:

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

**Calculate and display the average age of all students.**

SELECT AVG(AGE) AS AVREAGE\_AGE FROM STUDENT;

avreage\_age

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

21.6250000000000000

(1 row)

# QUERY 8:

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

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

SELECT STUDENT\_NAME FROM STUDENT

WHERE EMAIL LIKE '%example.com';

**student\_name**

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

SIVA

BASIL

SANDEEP

DINESH

EASHWAR

RAM

RAVI

VICTOR

(8 rows)

# QUESTIONS AND ANSWERS:

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

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

**primary key** is used to find the record uniquely from the table .

* + . Example: **STUDENT*\_*ID** in **STUDENT** table.

**Foreign key** is used to established the connection between the two tables.

* + Example:

**STUDENT*\_*ID** in **ENROLLMENT** table references **STUDENT*\_*ID** in **STUDENT**table.

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

**1.Varchar:**

* varchar datatype contain all characters.
* varchar variable length allocation

**2.Char:**

* char data type also contain all characters
* Char data type use fixed length allocation
  + Example: **STUDENT\_NAME** is VARCHAR because names vary in length.

**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.

. Example: **WHERE *email* LIKE*****'%example.com'***

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

**LIMIT:**

* Limit is used to give the limited rows according to number you enter

**OFFSET:**

* Offset is used for which row you want to start the record data
  + Example: **LIMIT 2 OFFSET 2** retrieves 2 rows starting from the 3rd row.

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

* UPDATE statement modifies existing records in a table based on specified conditions
* Example: **UPDATESTUDENTSETstatus *=* ‘AWARDED’WHERE** ...
* Updates the status of a student based on certain criteria.

**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.
* Example: LEFT JOIN ENROLLMENT ON C.COURSES\_ID = E.COURSE\_ID retrieves courses with the count of students enrolled.

**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.

**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 STUDENT** 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 STUDENT\_EMAIL ON STUDENT(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.
* Stores a query definition but not data itself. Simplifies complex queries, provides data security.