```
CREATE DATABASE UNIVERSITY_DB

CREATE TABLE STUDENTS (
        student_id INT PRIMARY KEY,
        student_name VARCHAR(50) ,
        AGE INT,
        EMAIL VARCHAR(50),
        frontend_mark INT,
        backend_mark INT,
        status VARCHAR(255)
)

CREATE TABLE COURSES(
        COURSE_ID INT PRIMARY KEY,
        COURSE_NAME VARCHAR(255),
        CREDITS INT
)

CREATE TABLE ENROLLMENT(
        ENROLLMENT_ID INT PRIMARY KEY,
        STUDENT_ID INT REFERENCES STUDENTS(student_id),
        COURSE_ID INT REFERENCES COURSES(COURSE_ID)
)



INSERT INTO
STUDENTS(STUDENT_ID,STUDENT_NAME,AGE,EMAIL,FRONTEND_MARK,BACKEND_MARK)
VALUES
(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@example.com',46,42)

INSERT INTO COURSES(COURSE_ID,COURSE_NAME,CREDITS)
VALUES
(1,'Next.js',3),
(2,'React.js',4),
(3,'Databases',3),
(4,'Prisma',3)

INSERT INTO ENROLLMENT (ENROLLMENT_ID,STUDENT_ID,COURSE_ID)
VALUES
(1,1,1),
(2,1,2),
(3,2,1),
(4,3,2)

SELECT * FROM STUDENTS

SELECT * FROM COURSES

SELECT * FROM ENROLLMENT

-- Query - 1

INSERT INTO
STUDENTS(STUDENT_ID,STUDENT_NAME,AGE,EMAIL,FRONTEND_MARK,BACKEND_MARK)
```

```
VALUES
(7,'Ajith Kumar',23,'ajith619006@gmail.com',99,100)



-- Query - 2
-- Retrieve the names of all students who are enrolled in the course titled 'Next.js'.
-- Sample Output:
-- student_name
-- Alice
-- Bob



SELECT S.STUDENT_NAME FROM STUDENTS S
INNER JOIN ENROLLMENT E USING(STUDENT_ID)
INNER JOIN COURSES C USING(COURSE_ID)
WHERE C.COURSE_NAME = 'Next.js'

-- Query 3:
-- Update the status of the student with the
-- highest total (frontend_mark + backend_mark) mark to 'Awarded'



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 TOPPER
```

```
)

SELECT * FROM STUDENTS



-- Query 4:
-- Delete all courses that have no students enrolled.

DELETE FROM COURSES
WHERE COURSE_ID NOT IN
(SELECT C.COURSE_ID FROM STUDENTS S
INNER JOIN ENROLLMENT E USING(STUDENT_ID)
INNER JOIN COURSES C USING(COURSE_ID)
)
RETURNING *;

SELECT * FROM COURSES

-- Query 5:
-- Retrieve the names of students using a limit of 2, starting from the 3rd student.
-- Sample Output:
-- student_name
-- Charlie
-- David

SELECT STUDENT_NAME FROM STUDENTS
OFFSET 2
LIMIT 2

-- 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

```
SELECT C.COURSE_NAME, COUNT(C.COURSE_NAME) AS students_enrolled FROM COURSES C
INNER JOIN ENROLLMENT USING(COURSE_ID)
GROUP BY COURSE_NAME
```

-- Query 7:

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

-- Sample Output:

-- average_age

-- 22.2857142857142857

```
SELECT AVG(AGE) AS average_age FROM STUDENTS
```

-- Query 8:

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

-- Sample Output:

-- student_name

-- Alice

-- Bob

-- Charlie

-- `David

```
SELECT STUDENT_NAME FROM STUDENTS
WHERE EMAIL LIKE '%example.com%'
```

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

PRIMARY KEY :

-> is used to ensure data in the specific column is unique

-> It is a combination of UNIQUE and Not Null constraints

-> Only one primary key is allowed in a table

FOREIGN KEY:

-> is a column or group of columns in a relational database table that provides a link between data in two tables

-> It refers to the field in a table which is the primary key of another table.

-> More than one foreign key is allowed in a table.

-> It can contain duplicate values also contain NULL values

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

CHAR :

-> To store strings of fixed size

-> Fixed amount of storage, based on the size of the column

-> Pads spaces to the right when storing strings less than the fixed size length ,Better performance

VARCHAR :

-> To store strings of variable length

-> Varying amounts of storage space based on the size of the string stored.

-> No padding necessary because it is variable in size,Slightly poorer performance because length has to be accounted

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

WHERE :

-> specifies any conditions for the results set. To filter data

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

LIMIT :

-> The LIMIT clause allows you to specify the maximum number of rows to return from a query

-> It can be used with or without an ORDER BY clause to sort the rows before limiting them

OFFSET :

-> The OFFSET clause allows you to skip a certain number of rows before returning the result set

-> It can be used with the LIMIT clause to create pagination or batch processing

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

SYNTAX :

```
->  UPDATE table_name
        SET column1 = value1,
            column2 = value2,
            ...
        WHERE condition;
```

-> Specify the name of the table that you want to update data after the UPDATE keyword.

-> Specify columns and their new values after SET keyword. The columns that do not appear in the SET clause retain their original values.

-> Determine which rows to update in the condition of the WHERE clause

-- 6. What is the significance of the JOIN operation, and how does it work in PostgreSQL?

PostgreSQL JOIN models logical relationships when retrieving data from multiple normalized database tables.

Normalization is a technique for reducing data redundancy, improving efficiency, and speeding up queries in a database

-> Readability. Developers comprehend JOIN statements faster than nested sub-queries.

-> Reduced execution time. A JOIN statement that uses indexed columns executes faster.

-> Flexibility. A JOIN statement uses a single command to fetch and filter data from multiple tables

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

GROUP BY :

-> command is a SELECT statement clause that divides the query result into groups of rows,

-> usually to perform one or more aggregations on each group.

-> The SELECT statement returns one row per group

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

-> command is a SELECT statement clause that divides the query result into groups of rows

-> Use the AVG() function to calculate the average value in a set of values.

-> Use the COUNT() function to perform a count.

-> Use the SUM() function to calculate the total of values.

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

-> are powerful tools for improving database performance, but their efficient use requires careful consideration

-> creating data structures that allow the database engine to quickly locate and retrieve specific rows from a table

1. B-tree Index for Price Range Queries

2. Single-Column Indexing

3. Hash Index for Category-Based Equality Checks

4. GiST Index for Text Search on Product Descriptions

5. BRIN Index for Date-Based Range Queries:

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

A view is a result of a SQL query. The result look like a table, however this table is not physically present in the database,

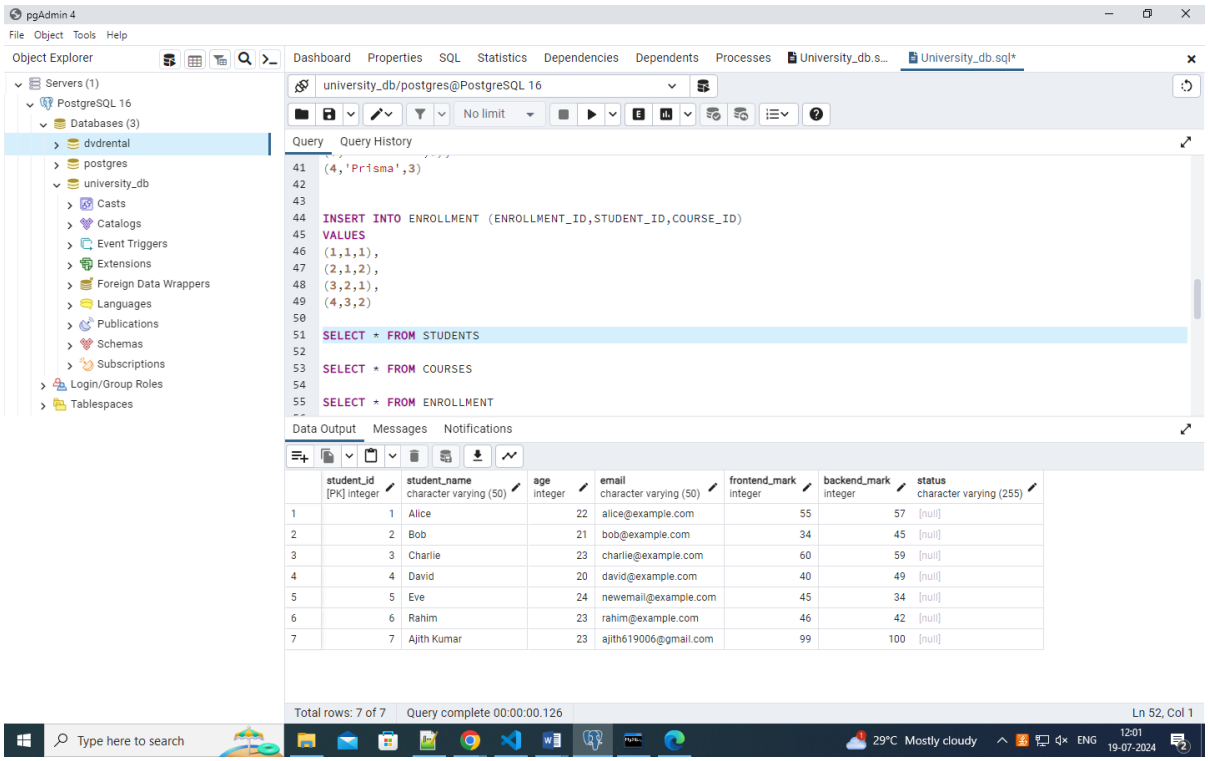
rather the data displayed as a view is fetched from the tables in database. This is why view is often referred as virtual table.

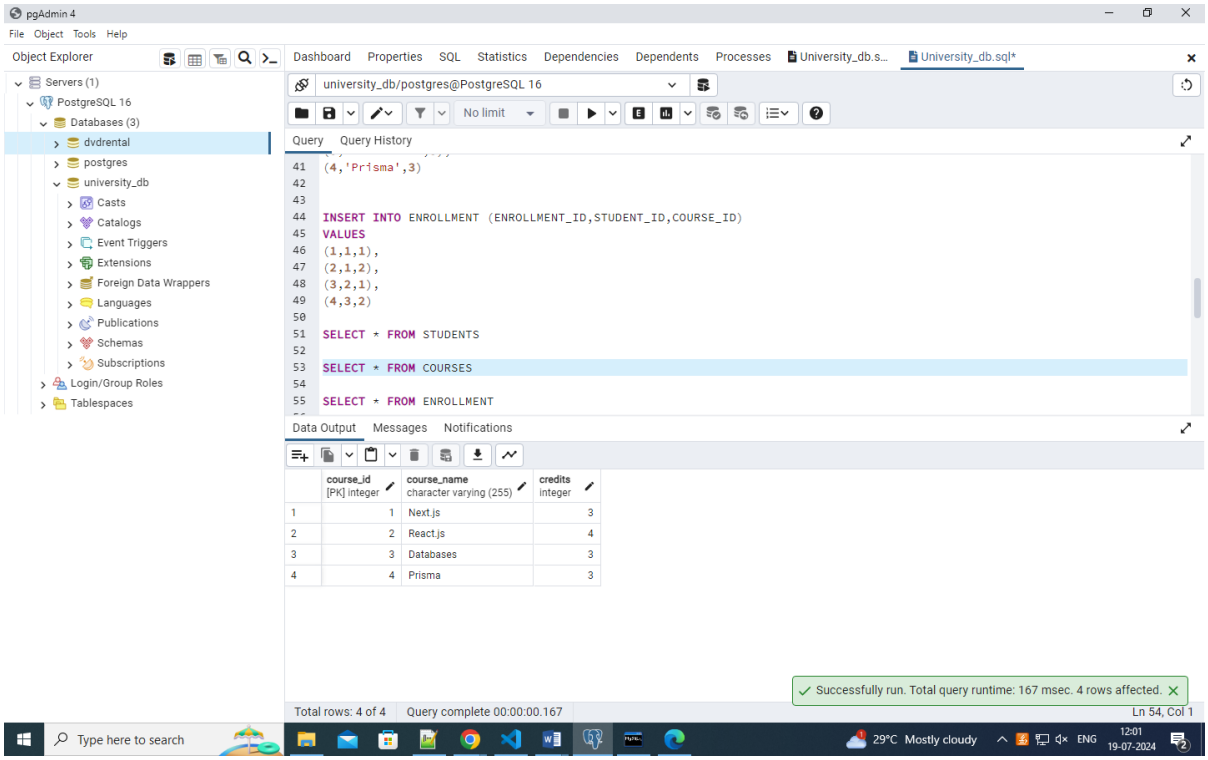
1) Simplifying complex queries

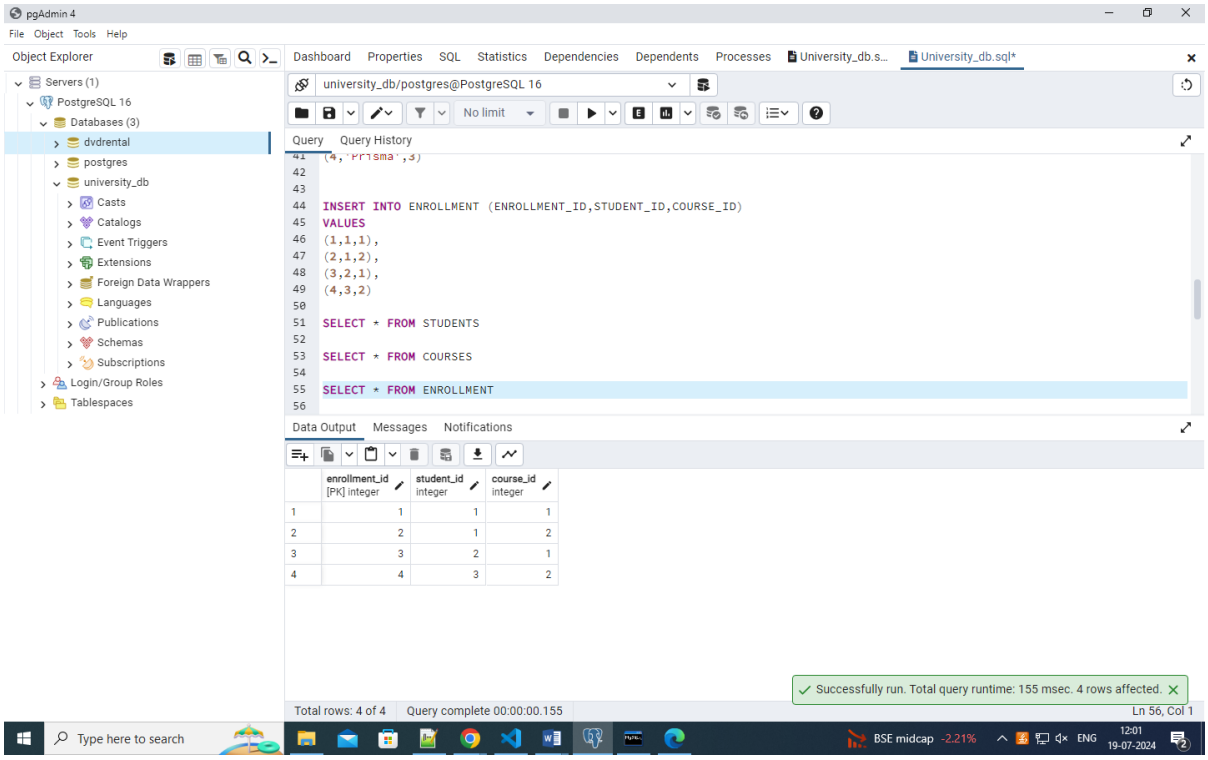
2) Security and access control

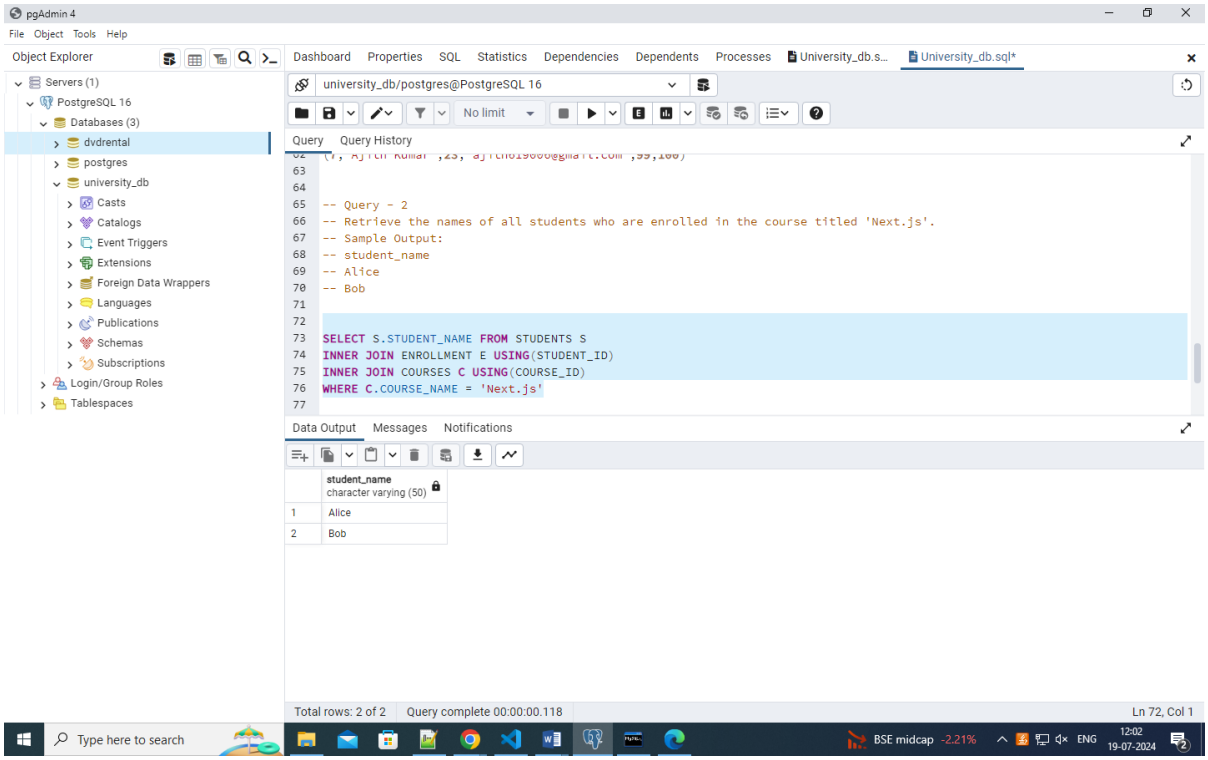
3) Logical data independence

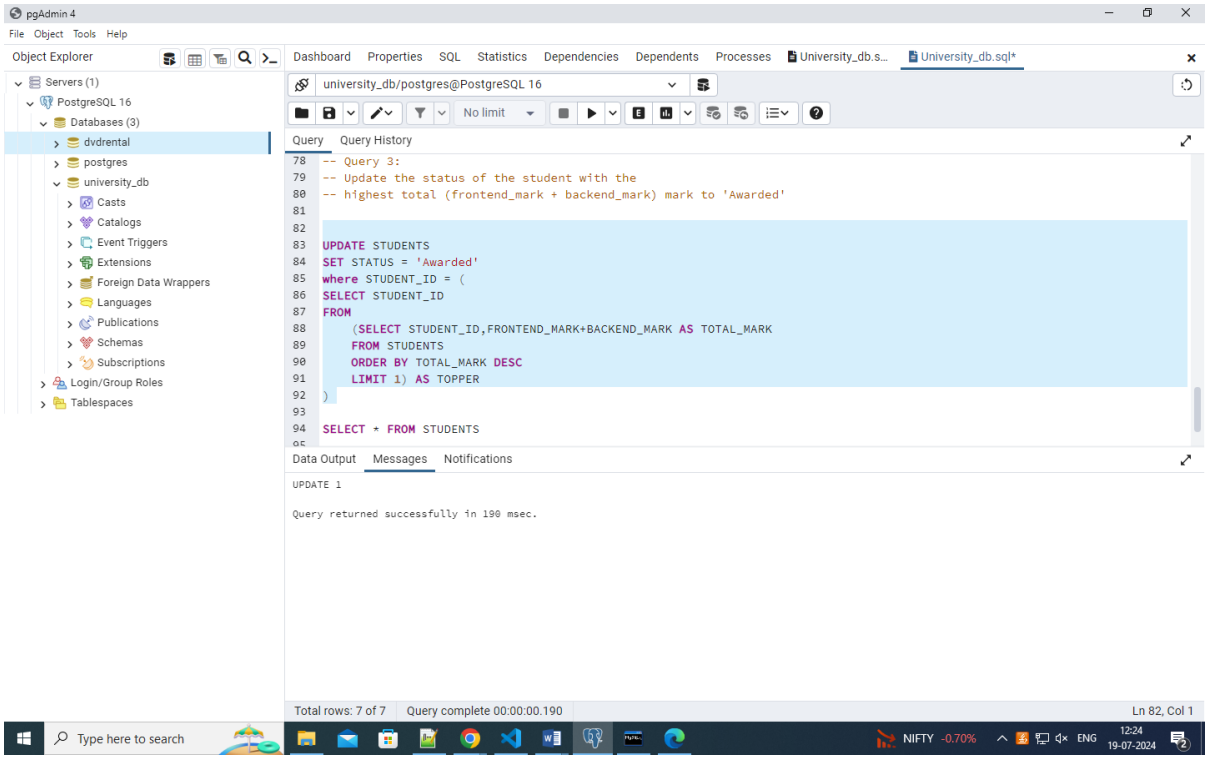
there are 5 types Basic PostgreSQL views , 2. Updatable views , 3. Materialized views , 4. Recursive views , 5. Managing views

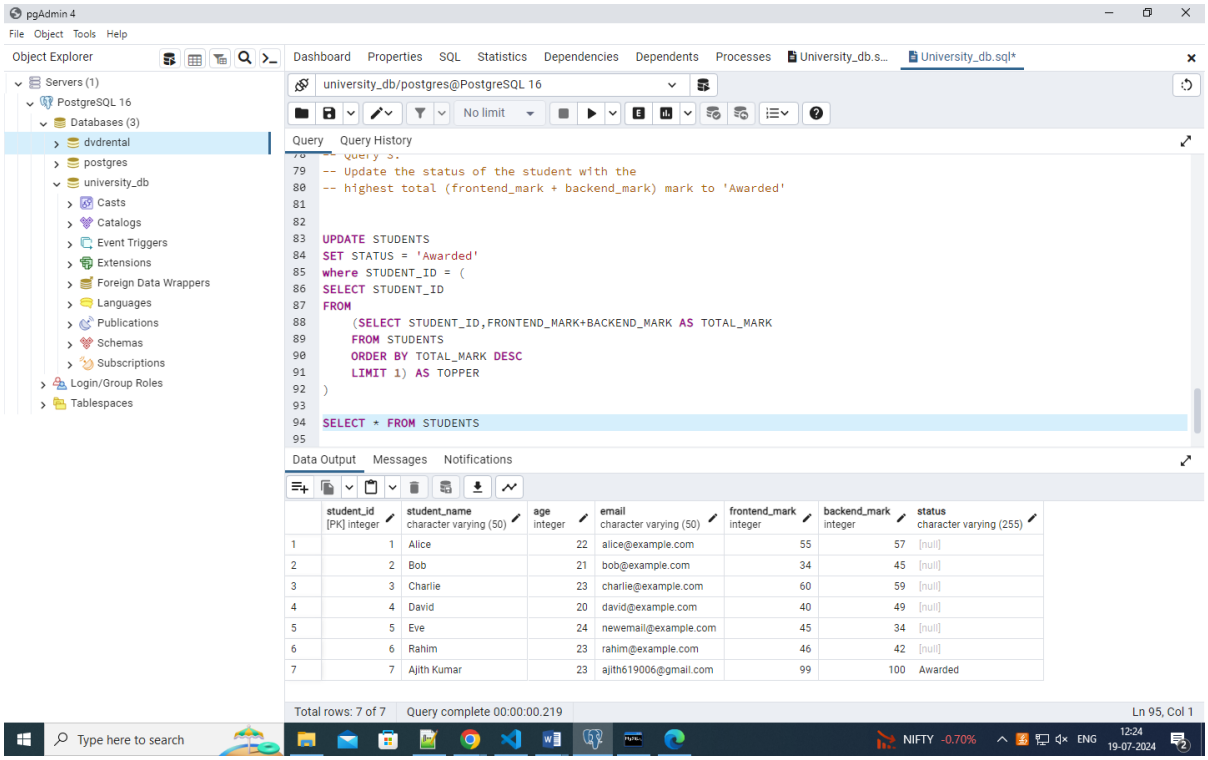


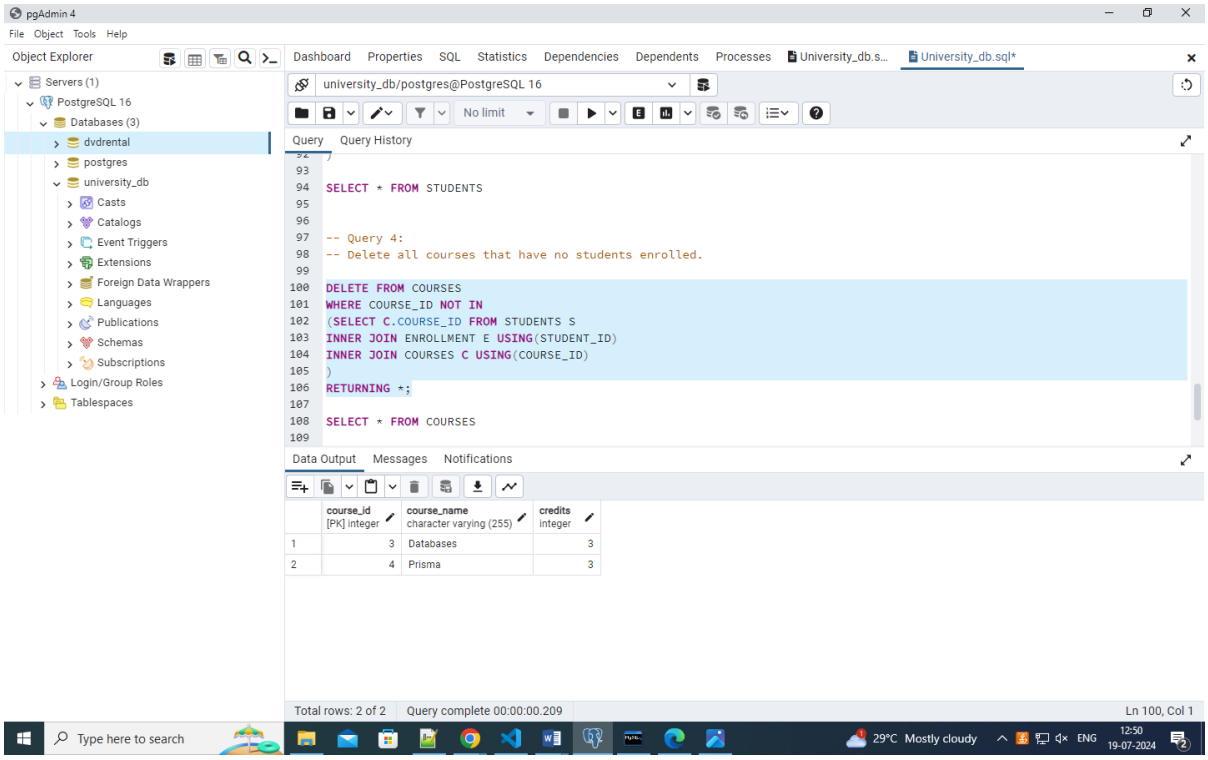

pgAdmin 4
File Object Tools Help
Object Explorer
Servers (1)
PostgreSQL 16
Databases (3)
dvdrental
postgres
university_db
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
Login/Group Roles
Tablespaces
Dashboard Properties SQL Statistics Dependencies Dependents Processes University_db.s... University_db.sql*
university_db/postgres@PostgreSQL 16
No limit
Query Query History
(4,'Prisma',3)
INSERT INTO ENROLLMENT (ENROLLMENT_ID,STUDENT_ID,COURSE_ID)
VALUES
(1,1,1),
(2,1,2),
(3,2,1),
(4,3,2)
SELECT * FROM STUDENTS
SELECT * FROM COURSES
SELECT * FROM ENROLLMENT
Data Output Messages Notifications
course_id [PK] integer
course_name character varying (255)
credits integer
1 1 Next.js 3
2 2 React.js 4
3 3 Databases 3
4 4 Prisma 3
Successfully run. Total query runtime: 167 msec. 4 rows affected.
Total rows: 4 of 4
Query complete 00:00:00.167
Ln 54, Col 1
Type here to search
29°C Mostly cloudy
ENG
12:01
19-07-2024

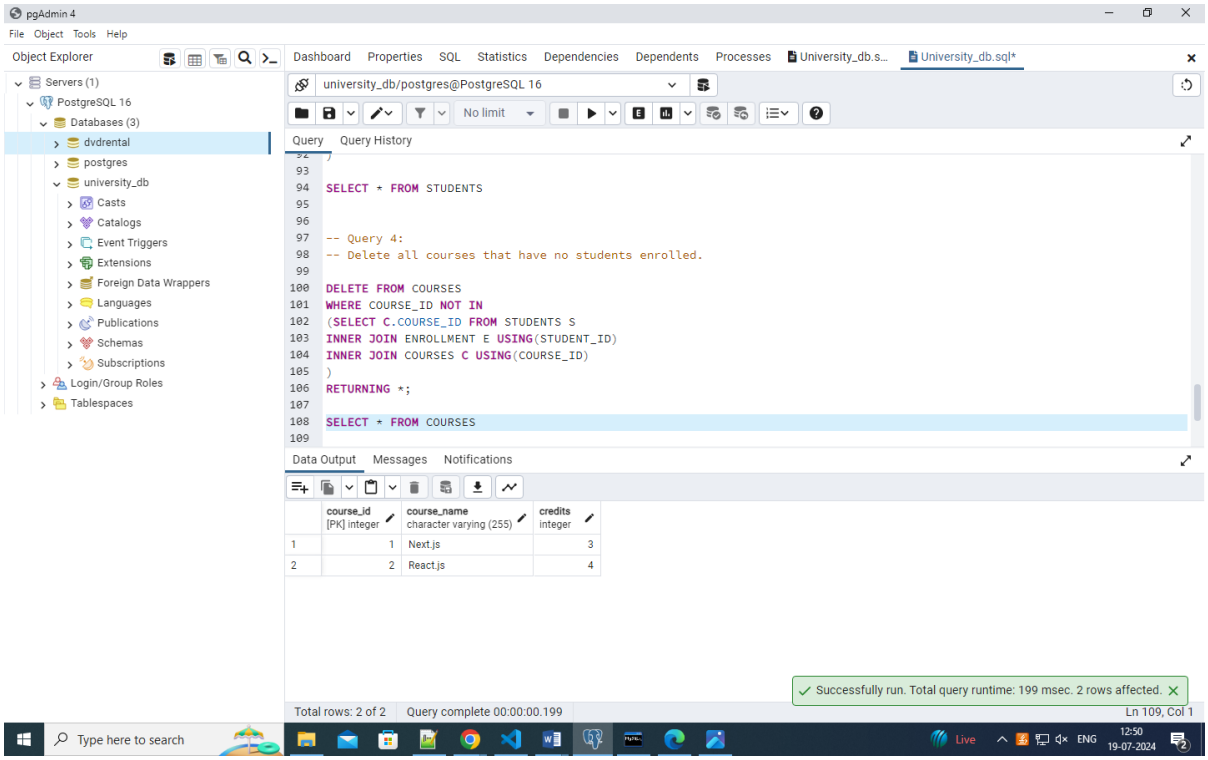

pgAdmin 4
File Object Tools Help
Object Explorer
Servers (1)
PostgreSQL 16
Databases (3)
dvdrental
postgres
university_db
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
Login/Group Roles
Tablespaces
Dashboard Properties SQL Statistics Dependencies Dependents Processes University_db.s... University_db.sql*
university_db/postgres@PostgreSQL 16
No limit
Query Query History
(4,'Prisma',3)
INSERT INTO ENROLLMENT (ENROLLMENT_ID,STUDENT_ID,COURSE_ID)
VALUES
(1,1,1),
(2,1,2),
(3,2,1),
(4,3,2)
SELECT * FROM STUDENTS
SELECT * FROM COURSES
SELECT * FROM ENROLLMENT
Data Output Messages Notifications
enrollment_id [PK] integer
student_id integer
course_id integer
1 1 1 1
2 2 1 2
3 3 2 1
4 4 3 2
Successfully run. Total query runtime: 155 msec. 4 rows affected.
Total rows: 4 of 4
Query complete 00:00:00.155
Ln 56, Col 1
Type here to search
BSE midcap -2.21%
ENG
12:01
19-07-2024

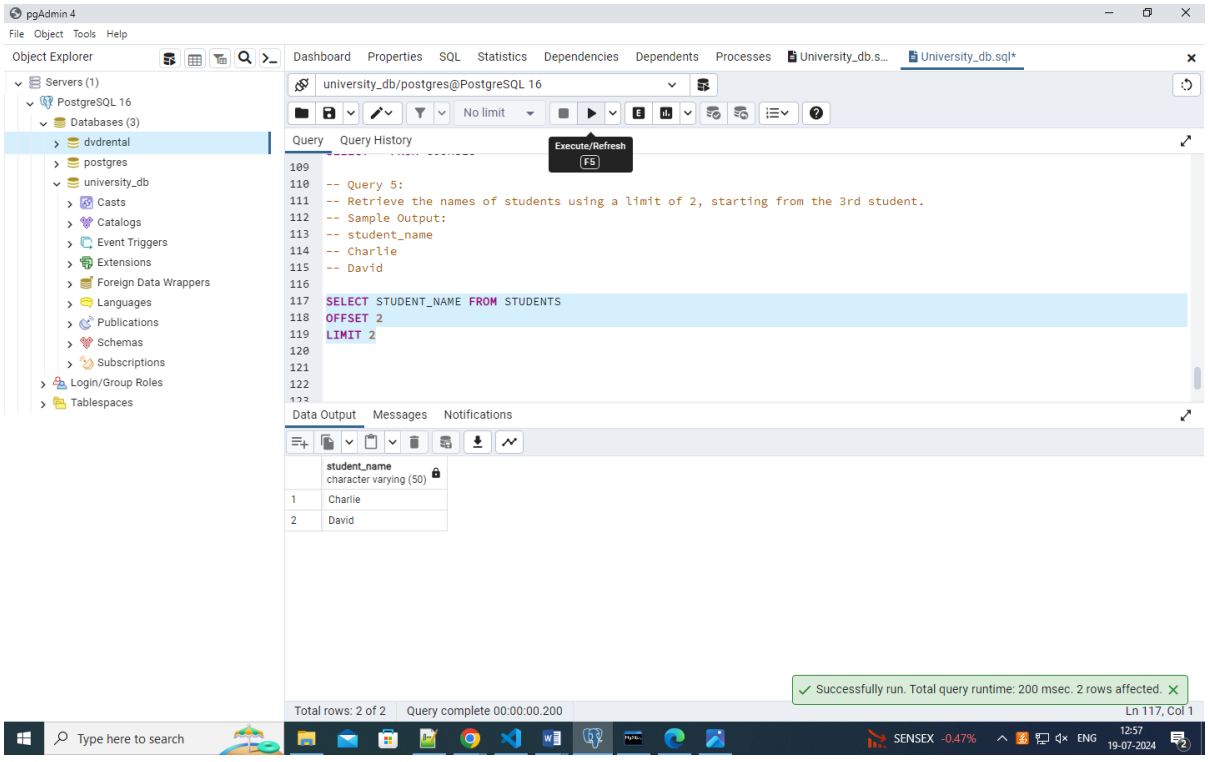

pgAdmin 4
File Object Tools Help
Object Explorer
Servers (1)
PostgreSQL 16
Databases (3)
dvdrental
postgres
university_db
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
Login/Group Roles
Tablespaces
Dashboard Properties SQL Statistics Dependencies Dependents Processes University_db.s... University_db.sql*
university_db/postgres@PostgreSQL 16
No limit
Query Query History
-- Query - 2
-- Retrieve the names of all students who are enrolled in the course titled 'Next.js'.
-- Sample Output:
-- student_name
-- Alice
-- Bob
SELECT S.STUDENT_NAME FROM STUDENTS S
INNER JOIN ENROLLMENT E USING(STUDENT_ID)
INNER JOIN COURSES C USING(COURSE_ID)
WHERE C.COURSE_NAME = 'Next.js'
Data Output Messages Notifications
student_name
character varying (50)
1 Alice
2 Bob
Total rows: 2 of 2 Query complete 00:00:00.118 Ln 72, Col 1
Type here to search
BSE midcap -2.21%
ENG
12:02
19-07-2024

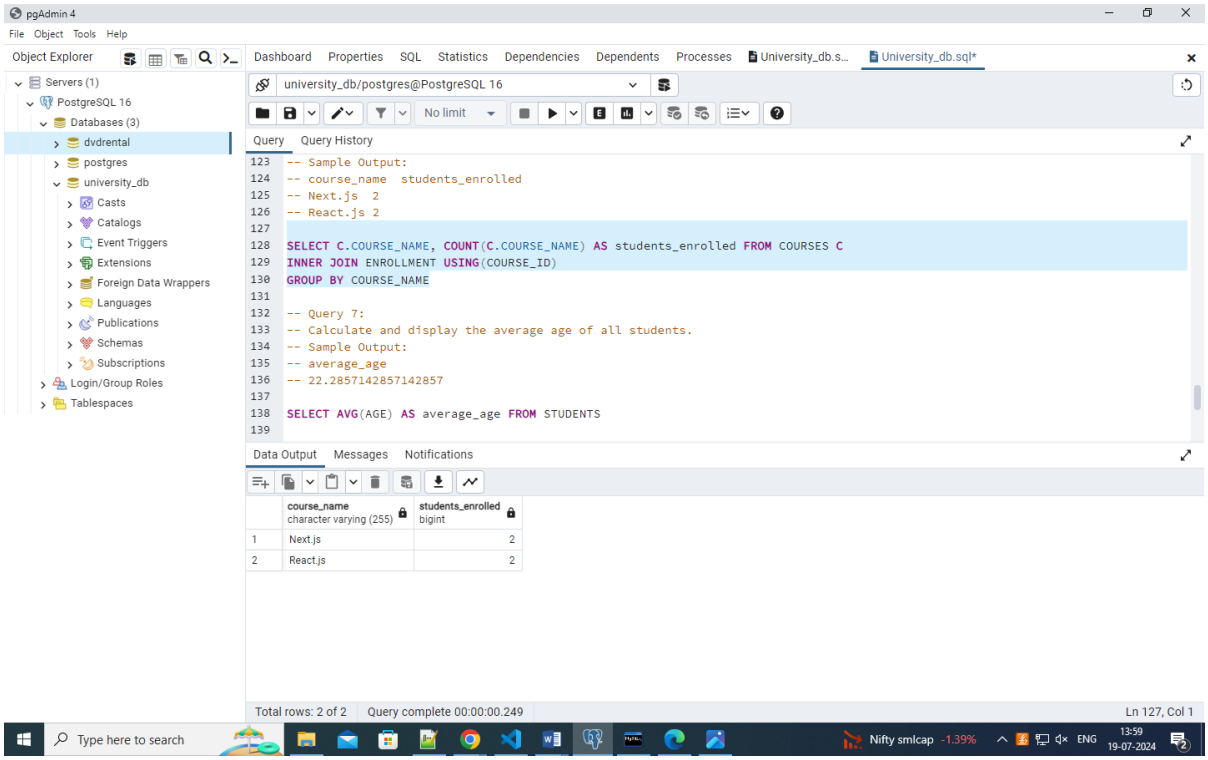

pgAdmin 4
File Object Tools Help
Object Explorer
Servers (1)
PostgreSQL 16
Databases (3)
dvdrental
postgres
university_db
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
Login/Group Roles
Tablespaces
Dashboard Properties SQL Statistics Dependencies Dependents Processes University_db.s... University_db.sql*
university_db/postgres@PostgreSQL 16
No limit
Query Query History
-- Query 3:
-- Update the status of the student with the
-- highest total (frontend_mark + backend_mark) mark to 'Awarded'
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 TOPPER
)
SELECT * FROM STUDENTS
Data Output Messages Notifications
UPDATE 1
Query returned successfully in 190 msec.
Total rows: 7 of 7 Query complete 00:00:00.190 Ln 82, Col 1
Type here to search
NIFTY -0.70%
ENG
12:24
19-07-2024

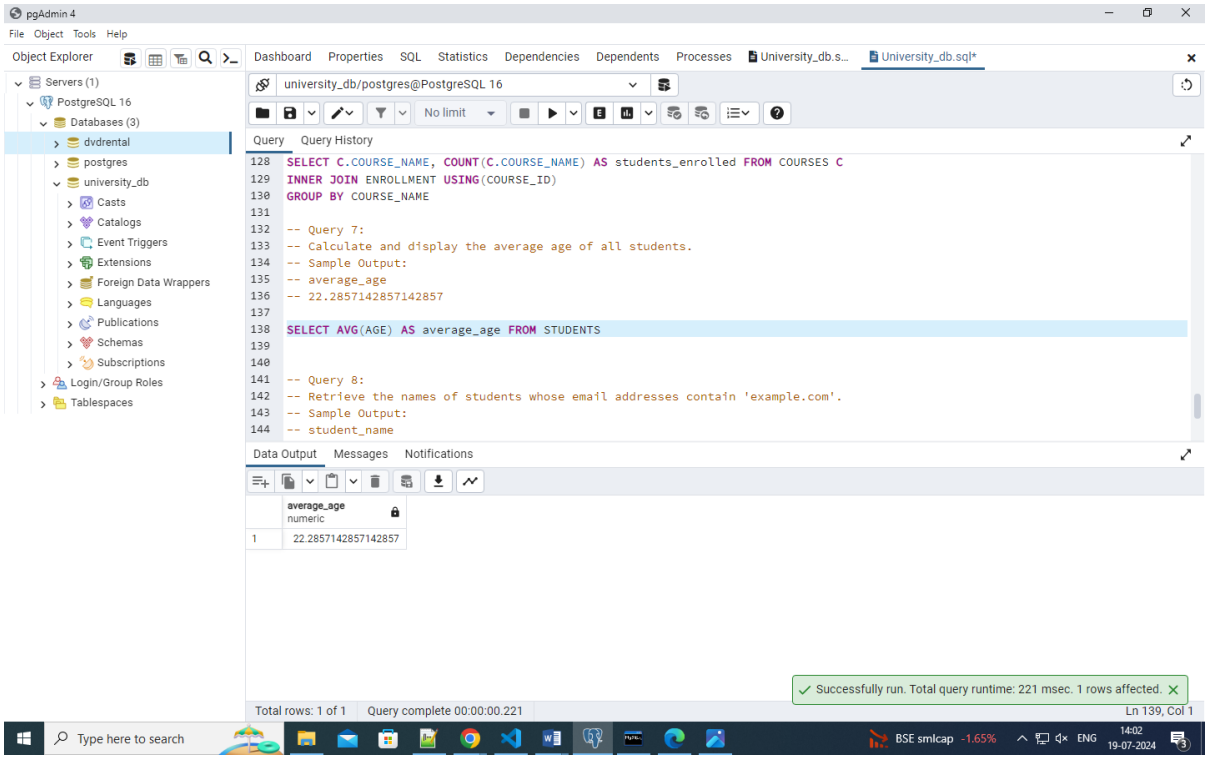

pgAdmin 4
File Object Tools Help
Object Explorer
Servers (1)
PostgreSQL 16
Databases (3)
dvdrental
postgres
university_db
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
Login/Group Roles
Tablespaces
Dashboard Properties SQL Statistics Dependencies Dependents Processes University_db.s... University_db.sql*
university_db/postgres@PostgreSQL 16
No limit
Query Query History
-- Query 3:
-- Update the status of the student with the
-- highest total (frontend_mark + backend_mark) mark to 'Awarded'
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 TOPPER
)
SELECT * FROM STUDENTS
Data Output Messages Notifications
student_id [PK] integer
student_name character varying (50)
age integer
email character varying (50)
frontend_mark integer
backend_mark integer
status character varying (255)
1 1 Alice 22 alice@example.com 55 57 [null]
2 2 Bob 21 bob@example.com 34 45 [null]
3 3 Charlie 23 charlie@example.com 60 59 [null]
4 4 David 20 david@example.com 40 49 [null]
5 5 Eve 24 newemail@example.com 45 34 [null]
6 6 Rahim 23 rahim@example.com 46 42 [null]
7 7 Ajith Kumar 23 ajith619006@gmail.com 99 100 Awarded
Total rows: 7 of 7
Query complete 00:00:00.219
Ln 95, Col 1
Type here to search
NIFTY -0.70%
ENG
12:24
19-07-2024

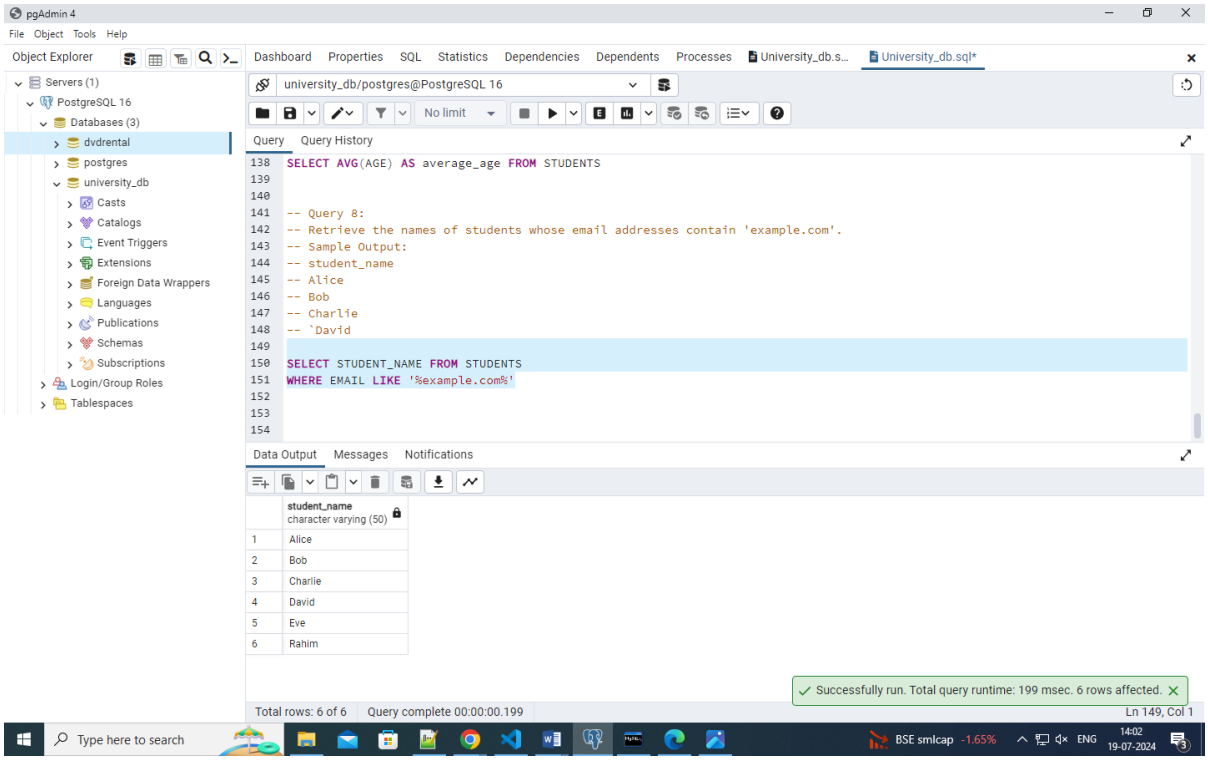

pgAdmin 4
File Object Tools Help
Object Explorer
Servers (1)
PostgreSQL 16
Databases (3)
dvdrental
postgres
university_db
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
Login/Group Roles
Tablespaces
Dashboard Properties SQL Statistics Dependencies Dependents Processes University_db.s... University_db.sql*
university_db/postgres@PostgreSQL 16
No limit
Query Query History
)
SELECT * FROM STUDENTS
-- Query 4:
-- Delete all courses that have no students enrolled.
DELETE FROM COURSES
WHERE COURSE_ID NOT IN
(SELECT C.COURSE_ID FROM STUDENTS S
INNER JOIN ENROLLMENT E USING(STUDENT_ID)
INNER JOIN COURSES C USING(COURSE_ID)
)
RETURNING *;
SELECT * FROM COURSES
Data Output Messages Notifications
course_id [PK] integer
course_name character varying (255)
credits integer
1 3 Databases 3
2 4 Prisma 3
Total rows: 2 of 2
Query complete 00:00:00.209
Ln 100, Col 1
Type here to search
29°C Mostly cloudy
ENG
12:50
19-07-2024


pgAdmin 4
File Object Tools Help
Object Explorer
Servers (1)
PostgreSQL 16
Databases (3)
dvdrental
postgres
university_db
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
Login/Group Roles
Tablespaces
Dashboard Properties SQL Statistics Dependencies Dependents Processes University_db.s... University_db.sql*
university_db/postgres@PostgreSQL 16
No limit
Query Query History
SELECT * FROM STUDENTS
-- Query 4:
-- Delete all courses that have no students enrolled.
DELETE FROM COURSES
WHERE COURSE_ID NOT IN
(SELECT C.COURSE_ID FROM STUDENTS S
INNER JOIN ENROLLMENT E USING(STUDENT_ID)
INNER JOIN COURSES C USING(COURSE_ID)
)
RETURNING *;
SELECT * FROM COURSES
Data Output Messages Notifications
course_id [PK] integer
course_name character varying (255)
credits integer
1 1 Next.js 3
2 2 React.js 4
Successfully run. Total query runtime: 199 msec. 2 rows affected.
Total rows: 2 of 2
Query complete 00:00:00.199
Ln 109, Col 1
Type here to search
Live
ENG
12:50
19-07-2024


pgAdmin 4
File Object Tools Help
Object Explorer
Servers (1)
PostgreSQL 16
Databases (3)
dvdrental
postgres
university_db
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
Login/Group Roles
Tablespaces
Dashboard Properties SQL Statistics Dependencies Dependents Processes University_db.s... University_db.sql*
university_db/postgres@PostgreSQL 16
No limit
Query Query History
Execute/Refresh
F5
-- Query 5:
-- Retrieve the names of students using a limit of 2, starting from the 3rd student.
-- Sample Output:
-- student_name
-- Charlie
-- David
SELECT STUDENT_NAME FROM STUDENTS
OFFSET 2
LIMIT 2
Data Output Messages Notifications
student_name character varying (50)
1 Charlie
2 David
Successfully run. Total query runtime: 200 msec. 2 rows affected.
Total rows: 2 of 2
Query complete 00:00:00.200
Ln 117, Col 1
Type here to search
SENSEX -0.47%
ENG
12:57
19-07-2024


pgAdmin 4
File Object Tools Help
Object Explorer
Dashboard Properties SQL Statistics Dependencies Dependents Processes University_db.s... University_db.sql*
university_db/postgres@PostgreSQL 16
No limit
Query Query History
123 -- Sample Output:
124 -- course_name students_enrolled
125 -- Next.js 2
126 -- React.js 2
127
128 SELECT C.COURSE_NAME, COUNT(C.COURSE_NAME) AS students_enrolled FROM COURSES C
129 INNER JOIN ENROLLMENT USING(COURSE_ID)
130 GROUP BY COURSE_NAME
131
132 -- Query 7:
133 -- Calculate and display the average age of all students.
134 -- Sample Output:
135 -- average_age
136 -- 22.2857142857142857
137
138 SELECT AVG(AGE) AS average_age FROM STUDENTS
139
Data Output Messages Notifications
course_name character varying (255)
students_enrolled bigint
1 Next.js 2
2 React.js 2
Total rows: 2 of 2 Query complete 00:00:00.249 Ln 127, Col 1
Type here to search
Nifty smlcap -1.39%
ENG 13:59 19-07-2024


pgAdmin 4
File Object Tools Help
Object Explorer
Dashboard Properties SQL Statistics Dependencies Dependents Processes University_db.s... University_db.sql*
university_db/postgres@PostgreSQL 16
No limit
Query Query History
128 SELECT C.COURSE_NAME, COUNT(C.COURSE_NAME) AS students_enrolled FROM COURSES C
129 INNER JOIN ENROLLMENT USING(COURSE_ID)
130 GROUP BY COURSE_NAME
131
132 -- Query 7:
133 -- Calculate and display the average age of all students.
134 -- Sample Output:
135 -- average_age
136 -- 22.2857142857142857
137
138 SELECT AVG(AGE) AS average_age FROM STUDENTS
139
140
141 -- Query 8:
142 -- Retrieve the names of students whose email addresses contain 'example.com'.
143 -- Sample Output:
144 -- student_name
Data Output Messages Notifications
average_age numeric
1 22.2857142857142857
Successfully run. Total query runtime: 221 msec. 1 rows affected.
Total rows: 1 of 1 Query complete 00:00:00.221 Ln 139, Col 1
Type here to search
BSE smlcap -1.65%
ENG 14:02 19-07-2024


pgAdmin 4
File Object Tools Help
Object Explorer
Servers (1)
PostgreSQL 16
Databases (3)
dvdrental
postgres
university_db
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
Login/Group Roles
Tablespaces
Dashboard Properties SQL Statistics Dependencies Dependents Processes University_db.s... University_db.sql*
university_db/postgres@PostgreSQL 16
No limit
Query Query History
138 SELECT AVG(AGE) AS average_age FROM STUDENTS
139
140
141 -- Query 8:
142 -- Retrieve the names of students whose email addresses contain 'example.com'.
143 -- Sample Output:
144 -- student_name
145 -- Alice
146 -- Bob
147 -- Charlie
148 -- `David
149
150 SELECT STUDENT_NAME FROM STUDENTS
151 WHERE EMAIL LIKE '%example.com%'
152
153
154
Data Output Messages Notifications
student_name character varying (50)
1 Alice
2 Bob
3 Charlie
4 David
5 Eve
6 Rahim
Successfully run. Total query runtime: 199 msec. 6 rows affected.
Total rows: 6 of 6 Query complete 00:00:00.199
Ln 149, Col 1
Type here to search
BSE smlcap -1.65%
ENG
14:02
19-07-2024