

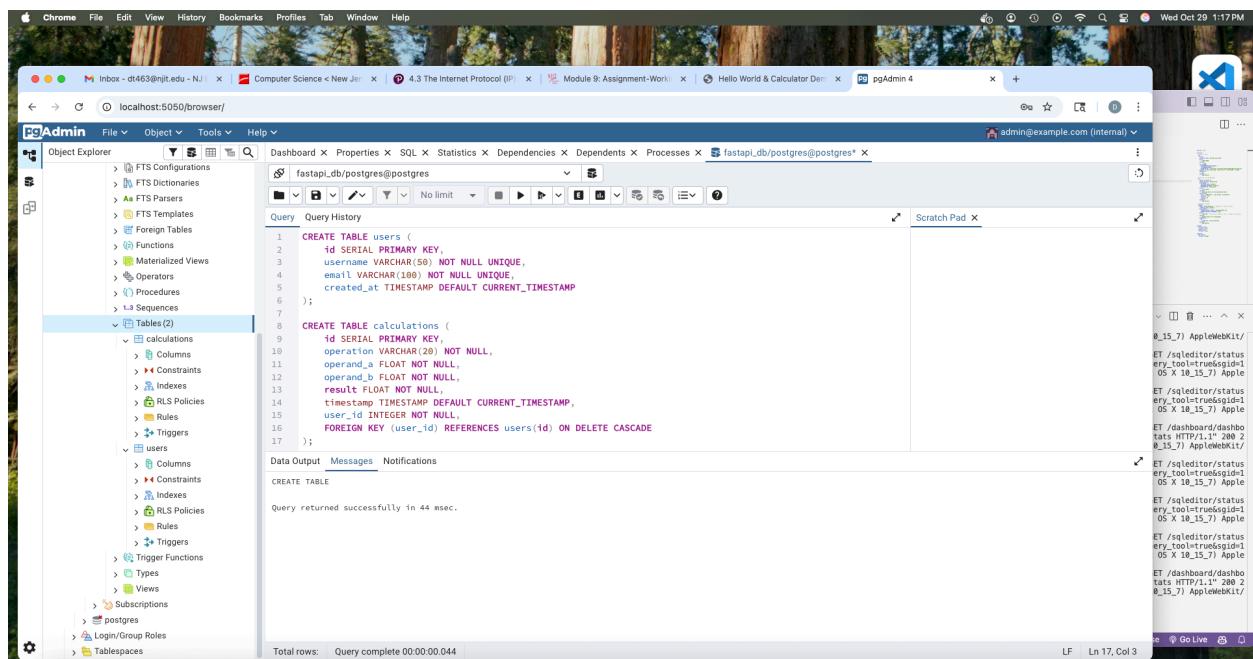
SQL Execution Documentation

(A) Create Tables

Query:

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    email VARCHAR(100) NOT NULL UNIQUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
CREATE TABLE calculations (
    id SERIAL PRIMARY KEY,
    operation VARCHAR(20) NOT NULL,
    operand_a FLOAT NOT NULL,
    operand_b FLOAT NOT NULL,
    result FLOAT NOT NULL,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    user_id INTEGER NOT NULL,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);
```



Screenshot Caption:

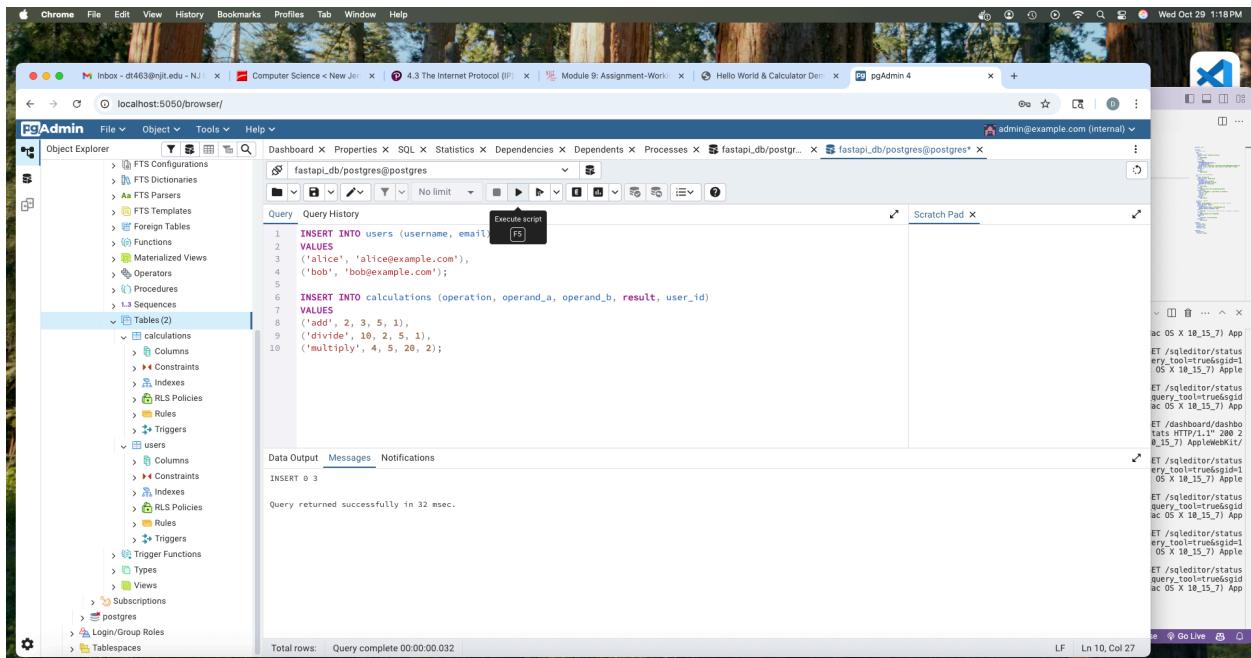
Tables `users` and `calculations` created successfully in pgAdmin. The output confirms structure and foreign key relationship.

(B) Insert Records

Query:

```
INSERT INTO users (username, email)
VALUES
('alice', 'alice@example.com'),
('bob', 'bob@example.com');
```

```
INSERT INTO calculations (operation, operand_a, operand_b,
result, user_id)
VALUES
('add', 2, 3, 5, 1),
('divide', 10, 2, 5, 1),
('multiply', 4, 5, 20, 2);
```



Screenshot Caption:

Records inserted into `users` and `calculations` tables. The output shows “Query returned successfully: X rows affected.”

(C) Query Data

1. Retrieve All Users

Query:

```
SELECT * FROM users;
```

The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'Tables' node, there are two entries: 'calculations' and 'users'. The 'users' table is selected, and its data is displayed in the Data Output tab. The table has four columns: id, username, email, and created_at. Two rows are present: one for user 'alice' and one for user 'bob'. The Data Output tab also shows the SQL query used to retrieve the data.

id	username	email	created_at
1	alice	alice@example.com	2025-10-29 17:18:09.725138
2	bob	bob@example.com	2025-10-29 17:18:09.725138

Screenshot Caption:

Displaying all users in the database. Shows `id`, `username`, `email`, and `created_at`.

2. Retrieve All Calculations

Query:

```
SELECT * FROM calculations;
```

```

-- Retrieve all users
SELECT * FROM users;

-- Retrieve all calculations
SELECT * FROM calculations;

-- Join users and calculations
SELECT u.username, c.operation, c.operand_a, c.operand_b, c.result
FROM calculations c
JOIN users u ON c.user_id = u.id;

```

	id [Pk] integer	operation character varying (20)	operand_a double precision	operand_b double precision	result double precision	timestamp timestamp without time zone	user_id integer
1	1	add	2	3	5	2025-10-29 17:18:09.725138	1
2	2	divide	10	2	5	2025-10-29 17:18:09.725138	1
3	3	multiply	4	5	20	2025-10-29 17:18:09.725138	2

Screenshot Caption:

Displaying all calculations. Includes operation type, operands, result, timestamp, and associated user ID.

3. Join Users and Calculations

Query:

```

SELECT u.username, c.operation, c.operand_a, c.operand_b,
c.result
FROM calculations c
JOIN users u ON c.user_id = u.id;

```

```

-- Retrieve all users
SELECT * FROM users;

-- Retrieve all calculations
SELECT * FROM calculations;

-- Join users and calculations
SELECT u.username, c.operation, c.operand_a, c.operand_b, c.result
FROM calculations c
JOIN users u ON c.user_id = u.id;

```

	username character varying (50)	operation character varying (20)	operand_a double precision	operand_b double precision	result double precision	timestamp
1	alice	add	2	3	5	2025-10-29 17:18:09.725138
2	alice	divide	10	2	5	2025-10-29 17:18:09.725138
3	bob	multiply	4	5	20	2025-10-29 17:18:09.725138

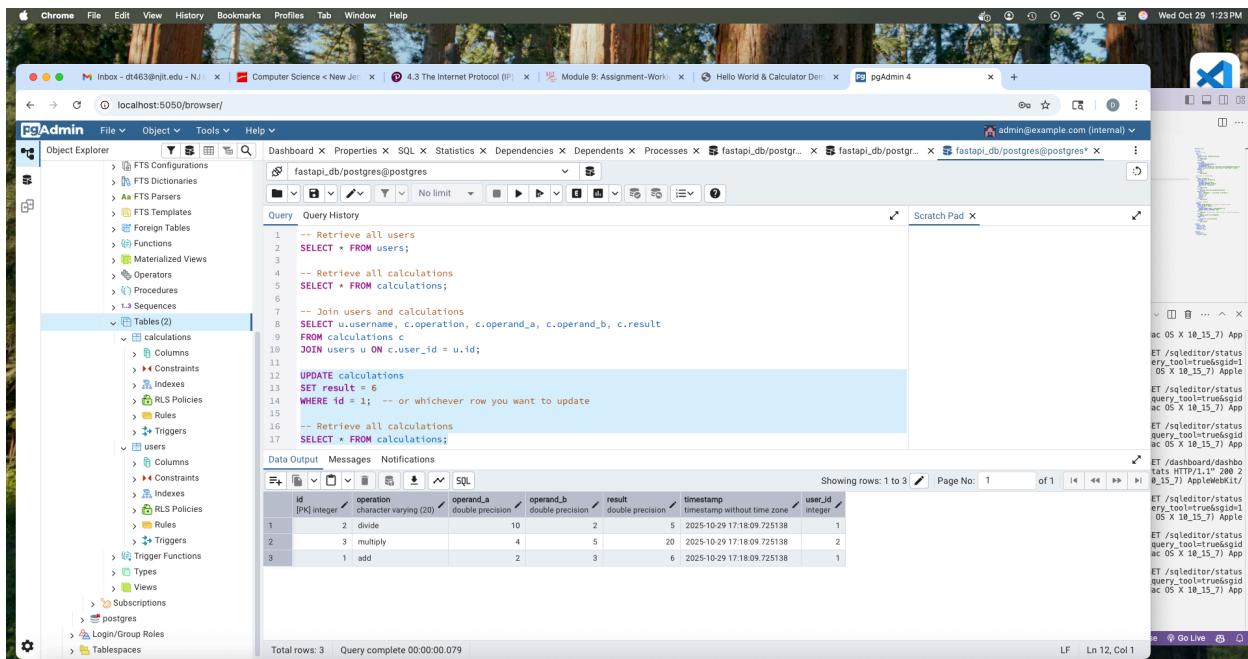
Screenshot Caption:

Joined `users` and `calculations` tables. Shows which user performed each operation and the result.

(D) Update a Record

Query:

```
UPDATE calculations
SET result = 6
WHERE id = 1;
```



The screenshot shows the pgAdmin 4 interface. The left sidebar displays the Object Explorer with the 'Tables' section selected, showing two tables: 'calculations' and 'users'. The main pane contains the SQL query:

```
-- Retrieve all users
SELECT * FROM users;
-- Retrieve all calculations
SELECT * FROM calculations;
-- Join users and calculations
SELECT u.username, c.operation, c.operand_a, c.operand_b, c.result
FROM calculations c
JOIN users u ON c.user_id = u.id;
UPDATE calculations
SET result = 6
WHERE id = 1; -- or whichever row you want to update
-- Retrieve all calculations
SELECT * FROM calculations;
```

The results pane shows a table with three rows of data:

ID	Operation	Operand A	Operand B	Result	Timestamp	User ID
1	divide	10	2	5	2025-10-29 17:18:09.725138	1
2	multiply	4	5	20	2025-10-29 17:18:09.725138	2
3	add	2	3	6	2025-10-29 17:18:09.725138	1

Log messages on the right side of the interface indicate successful execution of the query.

Screenshot Caption:

Updated calculation with `id = 1`. The result value changed from 5 to 6.

(E) Delete a Record

Query:

```
DELETE FROM calculations
WHERE id = 2;
```

The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'Tables' section, there is a 'calculations' table. A query window displays the following SQL code:

```

-- Join users and calculations
SELECT u.username, c.operation, c.operand_a, c.operand_b, c.result
FROM calculations c
JOIN users u ON c.user_id = u.id;
-- UPDATE calculations
SET result = 6
WHERE id = 1; -- or whichever row you want to update
-- Retrieve all calculations
SELECT * FROM calculations;
-- DELETE FROM calculations
WHERE id = 2; -- example record to remove
-- Retrieve all calculations
SELECT * FROM calculations;

```

The results of the final SELECT statement are shown in a Data Output grid:

	id [PK] integer	operation character varying(20)	operand_a double precision	operand_b double precision	result double precision	timestamp timestamp without time zone	user_id integer
1	3	multiply	4	5	20	2025-10-29 17:18:09.725138	2
2	1	add	2	3	6	2025-10-29 17:18:09.725138	1

The status bar at the bottom right indicates "Query complete 00:00:00.039".

Screenshot Caption:

Deleted calculation with `id = 2`. The output confirms successful deletion.

Reflection

In this assignment, I successfully set up a containerized FastAPI environment using Docker Compose, integrated PostgreSQL and pgAdmin, and executed raw SQL commands to manage relational data. I learned how foreign keys enforce relationships and how cascading deletes maintain data integrity. One challenge was resolving Docker volume conflicts, which I overcame by cleaning up old containers and pinning the PostgreSQL version. This experience strengthened my confidence in managing full-stack environments and debugging cross-service issues.