Web Development Advanced Concepts: Database Primer

Server-Side Data Management

Postgres Database in Container (Overview)

1. Create a separate directory "postgres-test" with a new Dockerfile"!

```
FROM postgres:latest
 ENV POSTGRES_USER postgres
 ENV POSTGRES_PASSWORD 12345
 ENV POSTGRES_DB postgres

EXPOSE 5432
```

Just Overview,
we go along step-by-step

2. Create an image "my-postgres-image"

```
docker build -t my-postgres-image .
```

3. Create and run a new container "my-postgres-container"

```
docker run -d -p 5432:5432 --name my-postgres-container my-postgres-image
```

4. Enter "my-postgres-container"

```
docker exec -it my-postgres-container bash
```

5. Connect to the database

```
psql postgresql://postgres:12345@127.0.0.1:5432/postgres
```

Server-Side Data Management

Postgres Database in Container

1. Create a separate directory "postgres-test" with a new Dockerfile"!

```
mkdir postgres-test
cd postgres-test
```

Open "postgres-test" in vs code and create a new Dockerfile"!

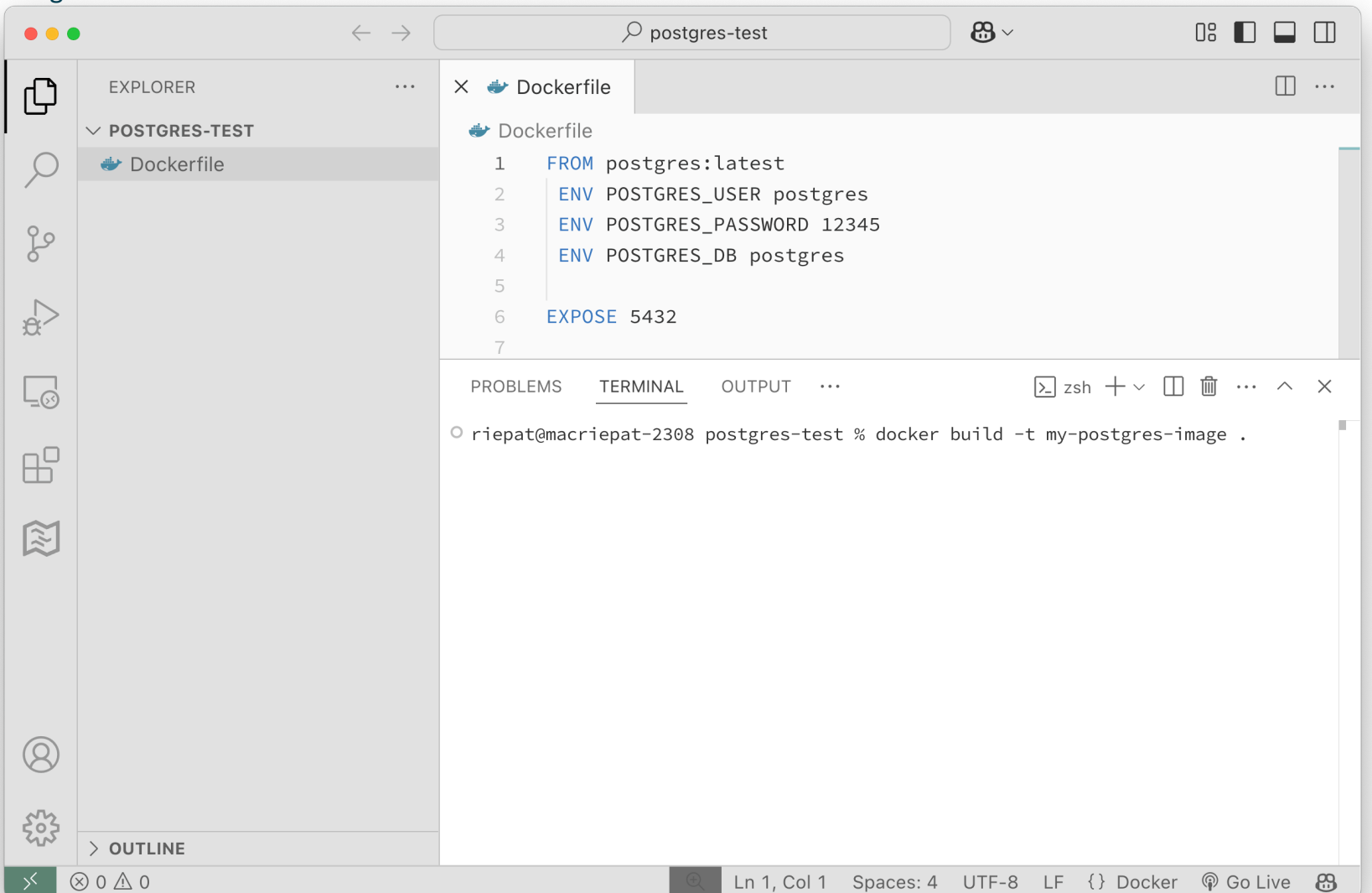
```
FROM postgres:latest
 ENV POSTGRES_USER postgres
 ENV POSTGRES_PASSWORD 12345
 ENV POSTGRES_DB postgres

EXPOSE 5432
```

```
docker build -t my-postgres-image .
```

Server-Side Data Management

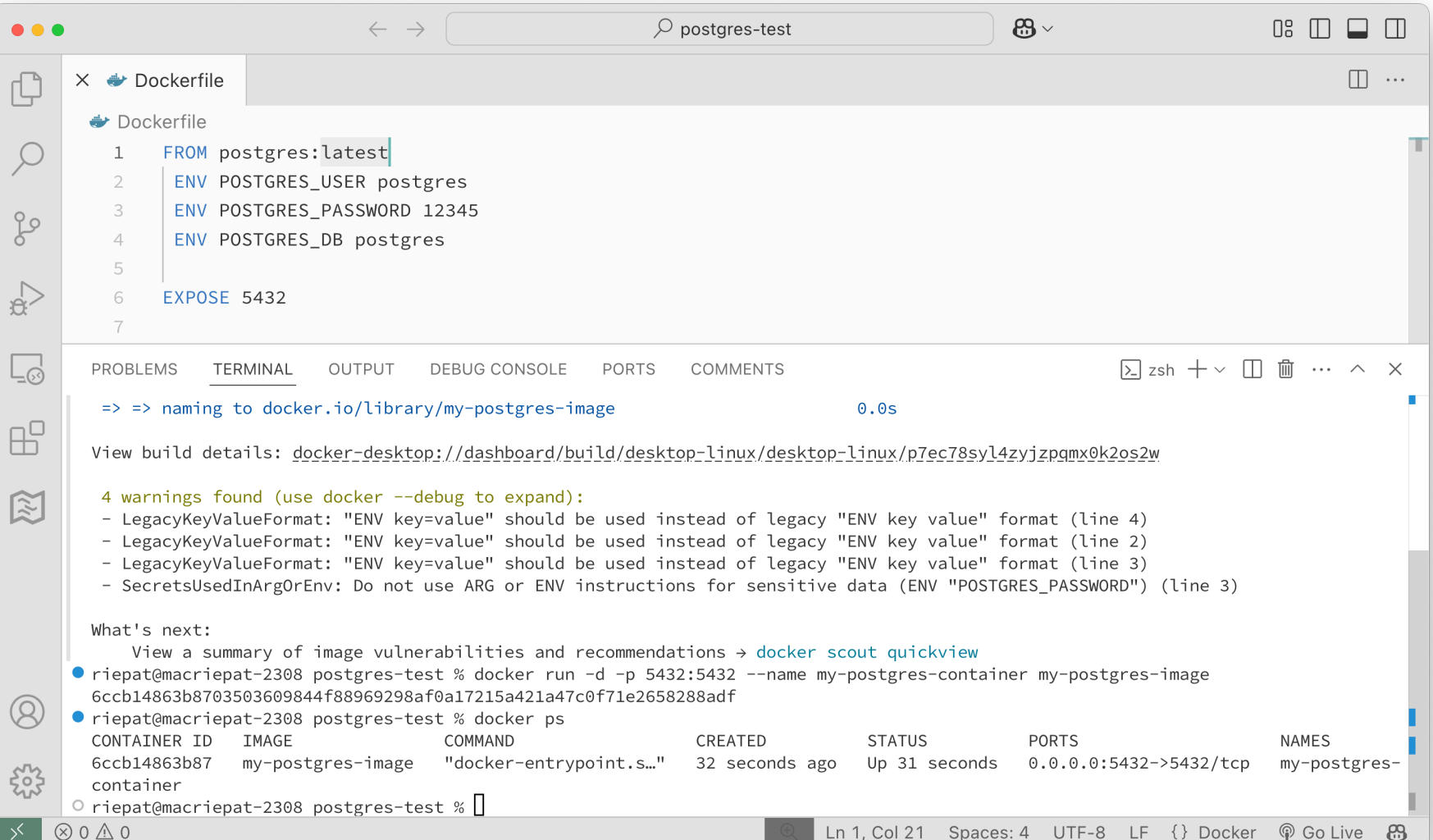
Postgres Database in Container




```
docker build -t my-postgres-image .
```

Server-Side Data Management

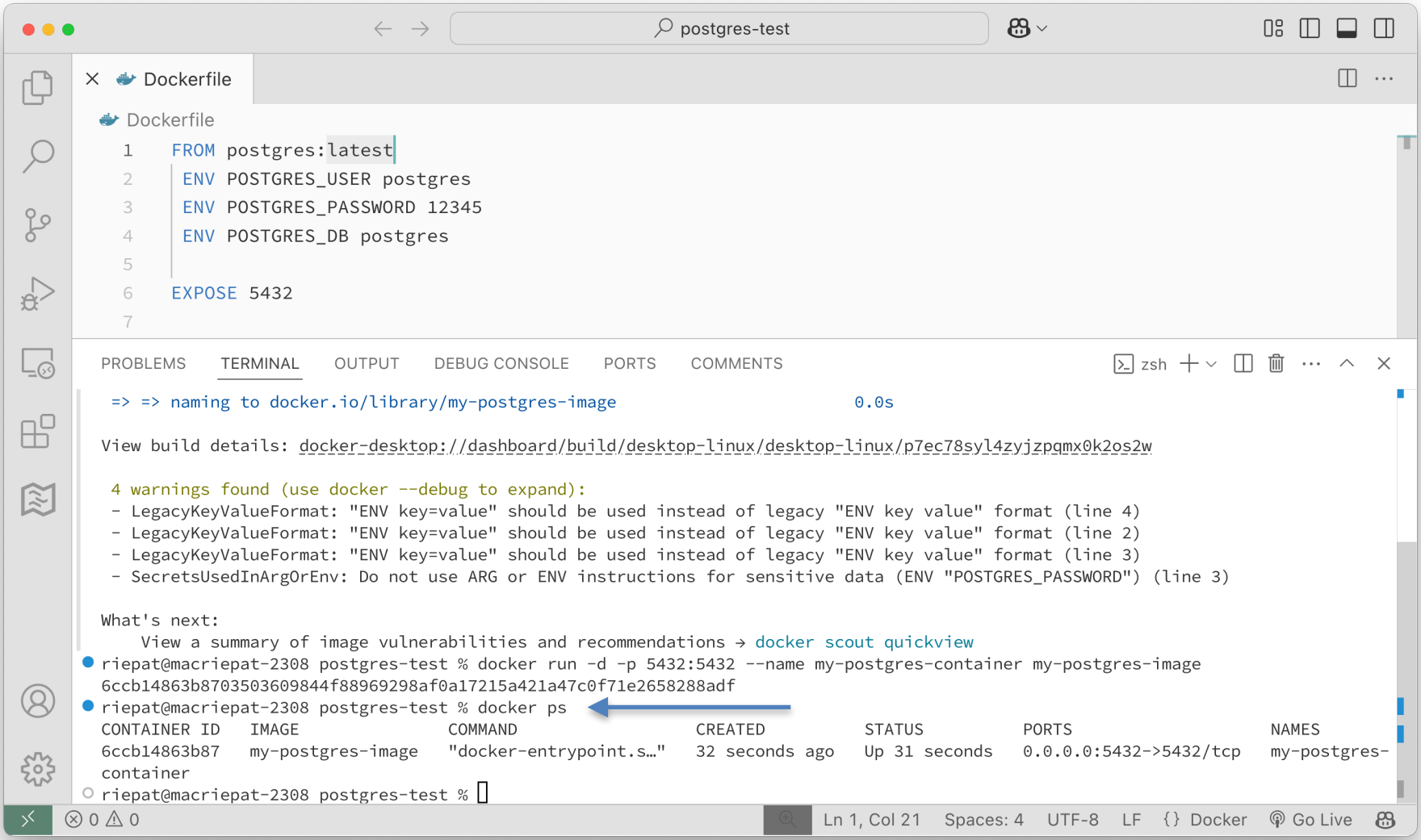
Postgres Database Operations




```
docker run -d -p 5432:5432 --name my-postgres-container my-postgres-image
```

Server-Side Data Management

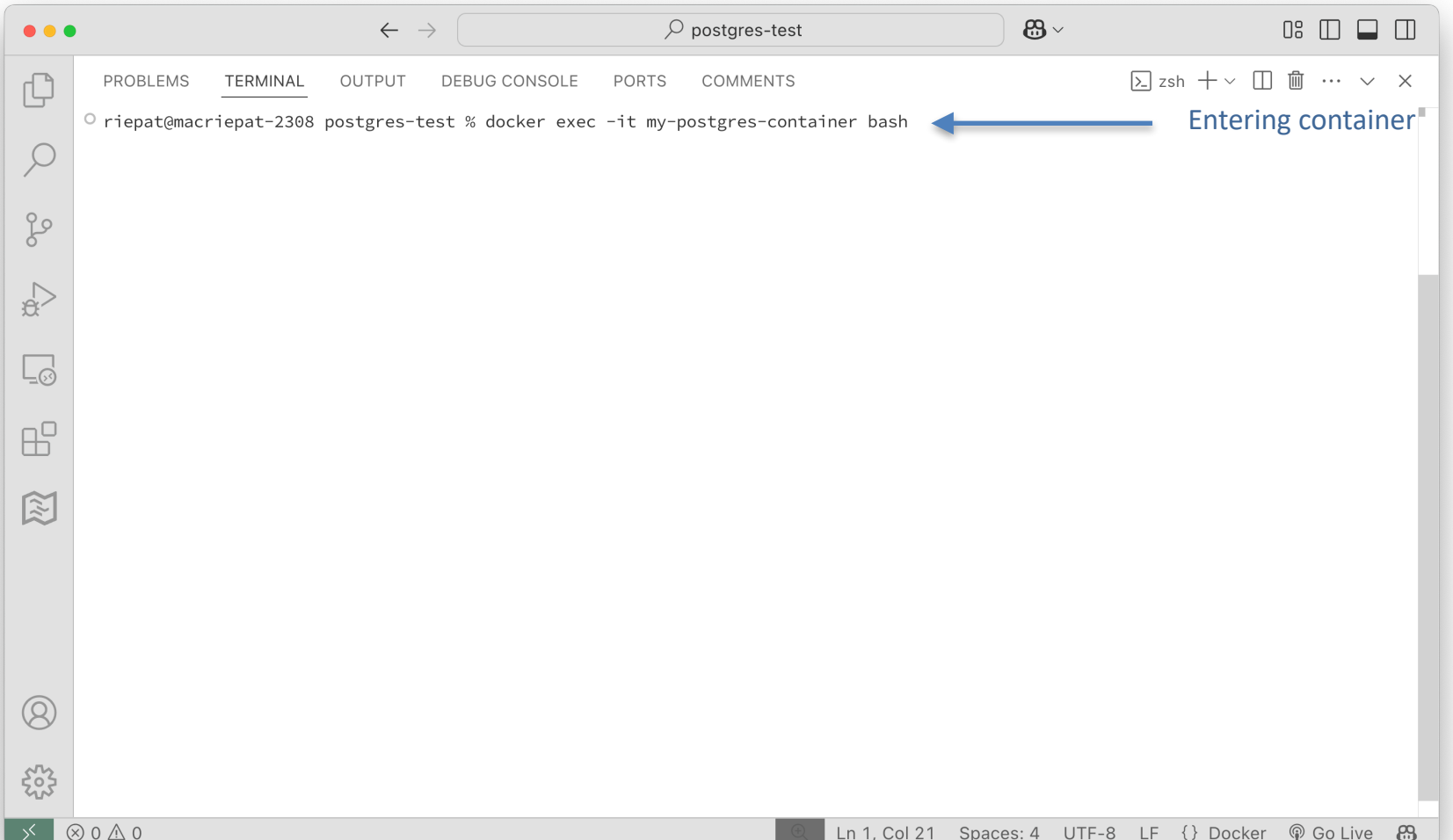
Postgres Database Operations



`docker ps`

Server-Side Data Management

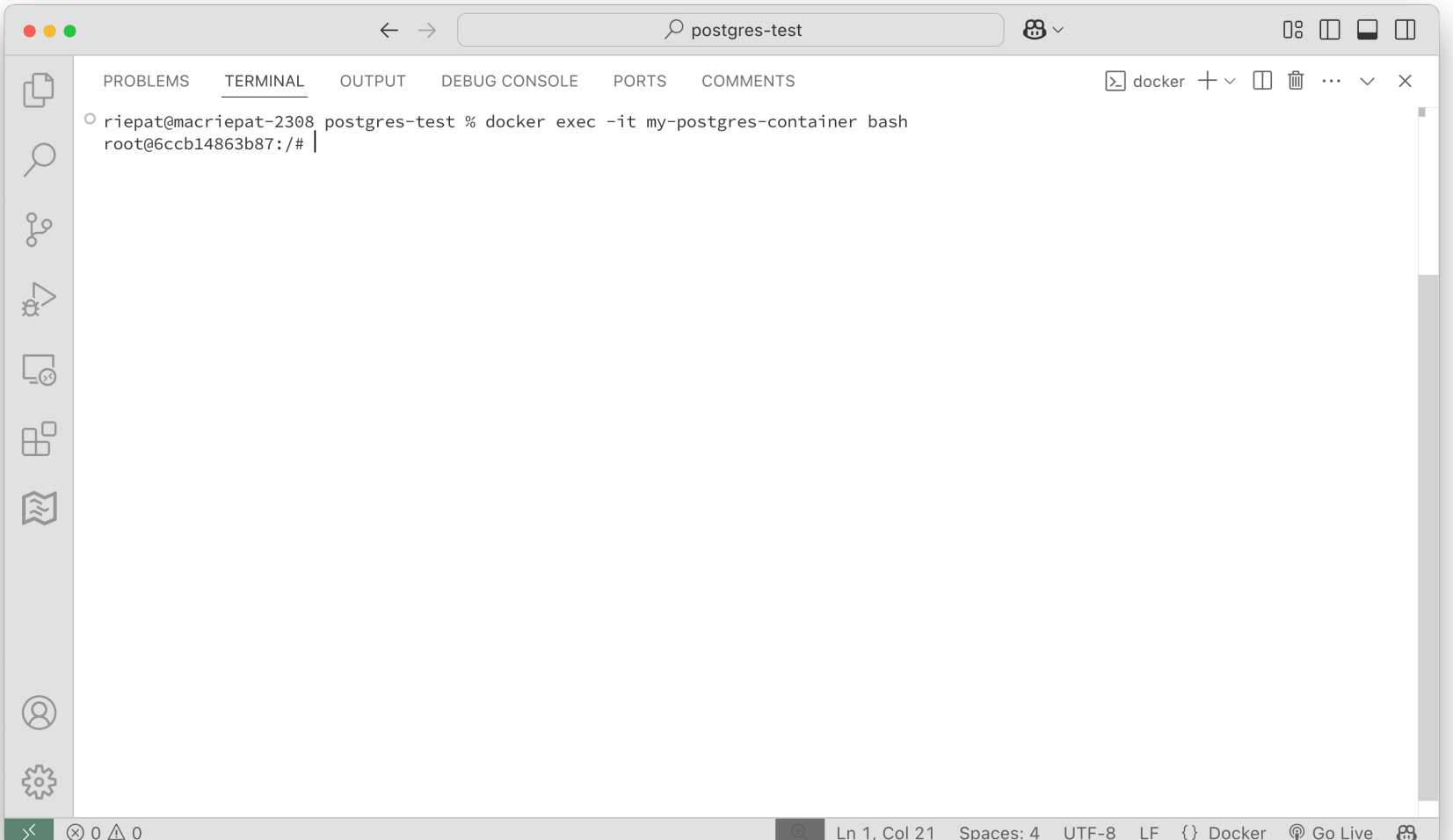
Postgres Database Operations




```
docker exec -it my-postgres-container bash
```

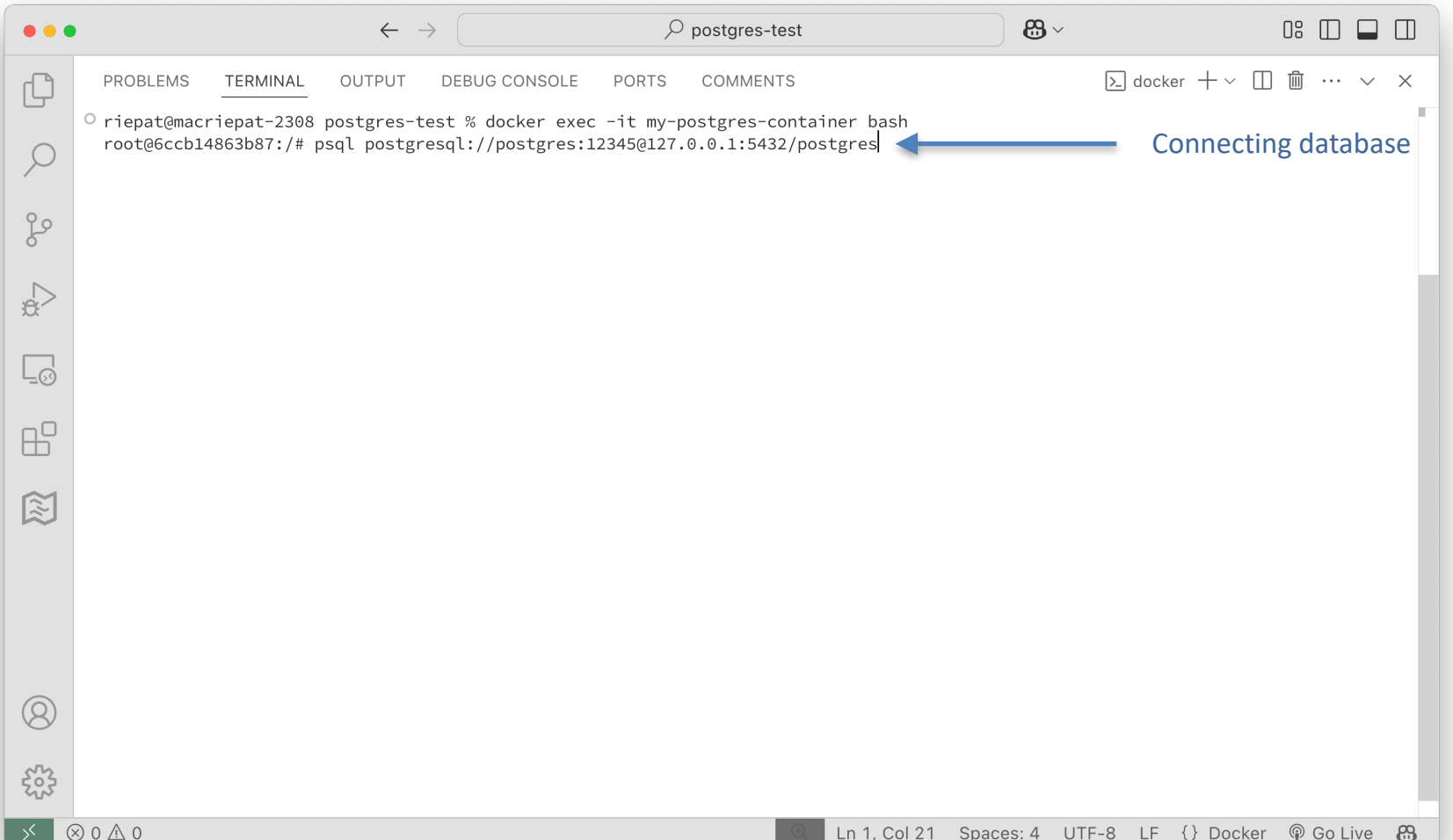
Server-Side Data Management

Postgres Database Operations



Server-Side Data Management

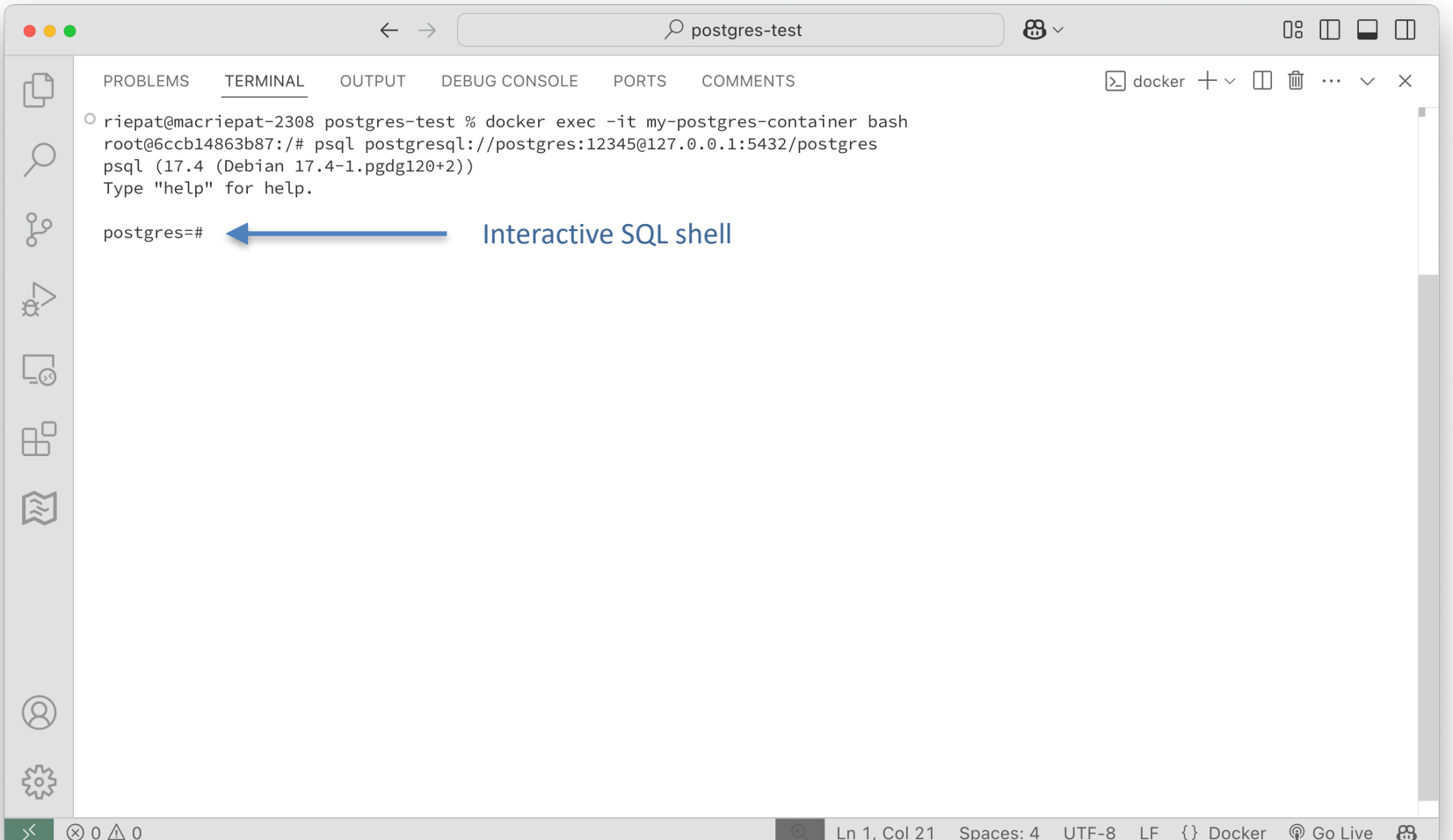
Postgres Database Operations




```
psql postgresql://postgres:12345@127.0.0.1:5432/postgres
```

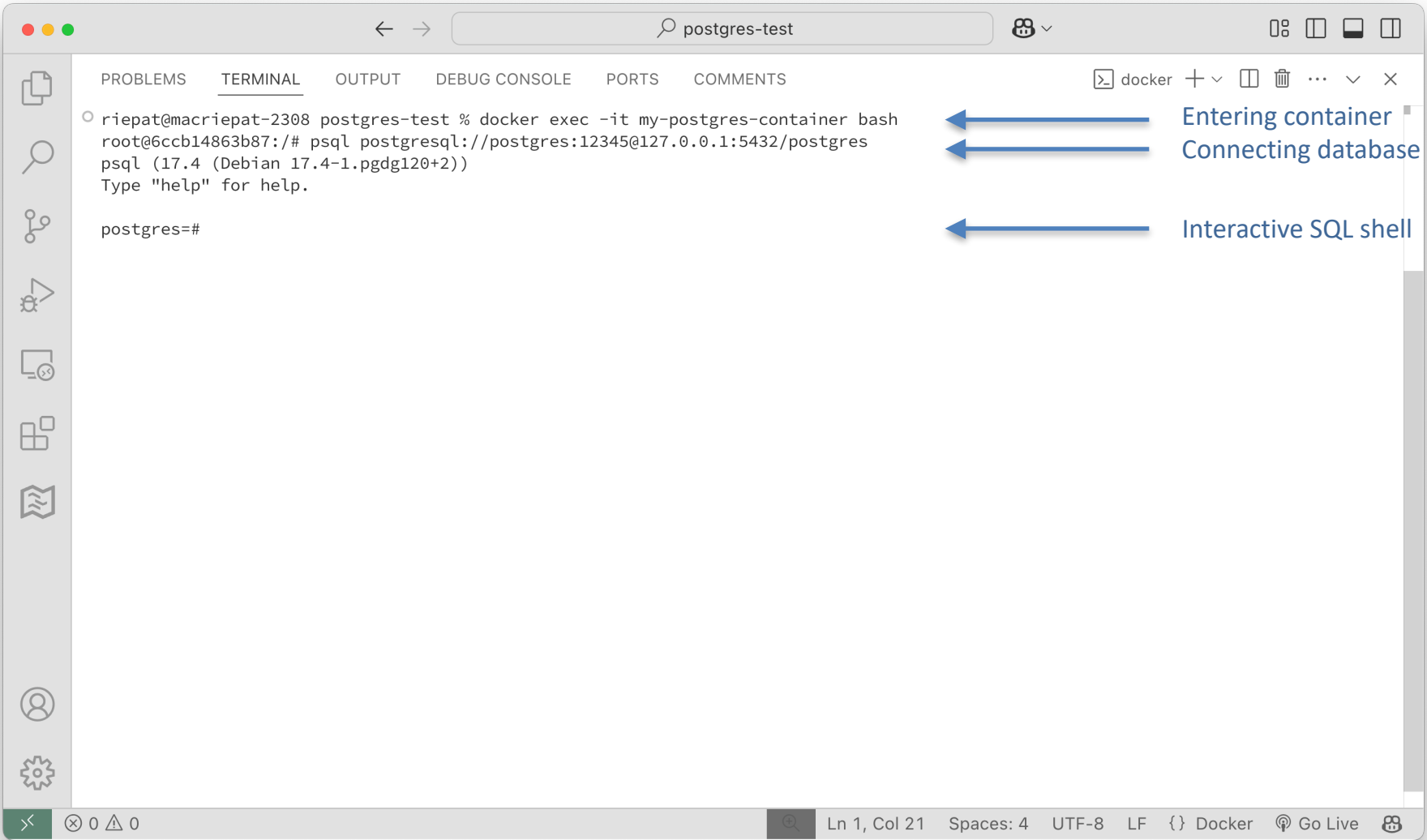
Server-Side Data Management

Postgres Database Operations



Server-Side Data Management

Postgres Database Operations



Server-Side Data Management

Postgres Database Operations

1. Create a table "employees" inside `psql` with the following scheme

```
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    hire_date DATE
);
```

SERIAL auto-increments the `id` field.
PRIMARY KEY enforces uniqueness.
VARCHAR(n) specifies variable-length strings.

UNIQUE ensures no duplicate email addresses.

(optional) Check existing tables

```
\dt
```

```
          List of relations
 Schema |   Name    | Type  |  Owner
--------+-----------+-------+----------
 public | employees | table | postgres
(1 row)
```

2. Insert "records"

```
INSERT INTO employees (first_name, last_name, email, hire_date)
VALUES ('John', 'Doe', 'john.doe@example.com', '2023-01-15');

INSERT INTO employees (first_name, last_name, email, hire_date)
VALUES ('Patrick', 'r', 'patrick.r@example.com', '2023-09-01');
```

Server-Side Data Management

Postgres Database Operations

3. Querying Data with SELECT

```
SELECT * FROM employees;
```

```
 id | first_name | last_name |        email         | hire_date
----+------------+-----------+----------------------+------------
  1 | John       | Doe       | john.doe@example.com | 2023-01-15
  2 | Patrick    | r         | patrick.r@example.com | 2023-09-01
(2 rows)
```

```
SELECT first_name, last_name FROM employees;
```

```
 first_name | last_name
------------+-----------
 John       | Doe
 Patrick    | r
(2 rows)
```

4. Updating records

```
UPDATE employees
SET email = 'john.newemail@example.com'
WHERE id = 1;
```

```
 id | first_name | last_name |           email            | hire_date
----+------------+-----------+----------------------------+------------
  2 | Patrick    | r         | patrick.r@example.com      | 2023-09-01
  1 | John       | Doe       | john.newemail@example.com | 2023-01-15
(2 rows)
```

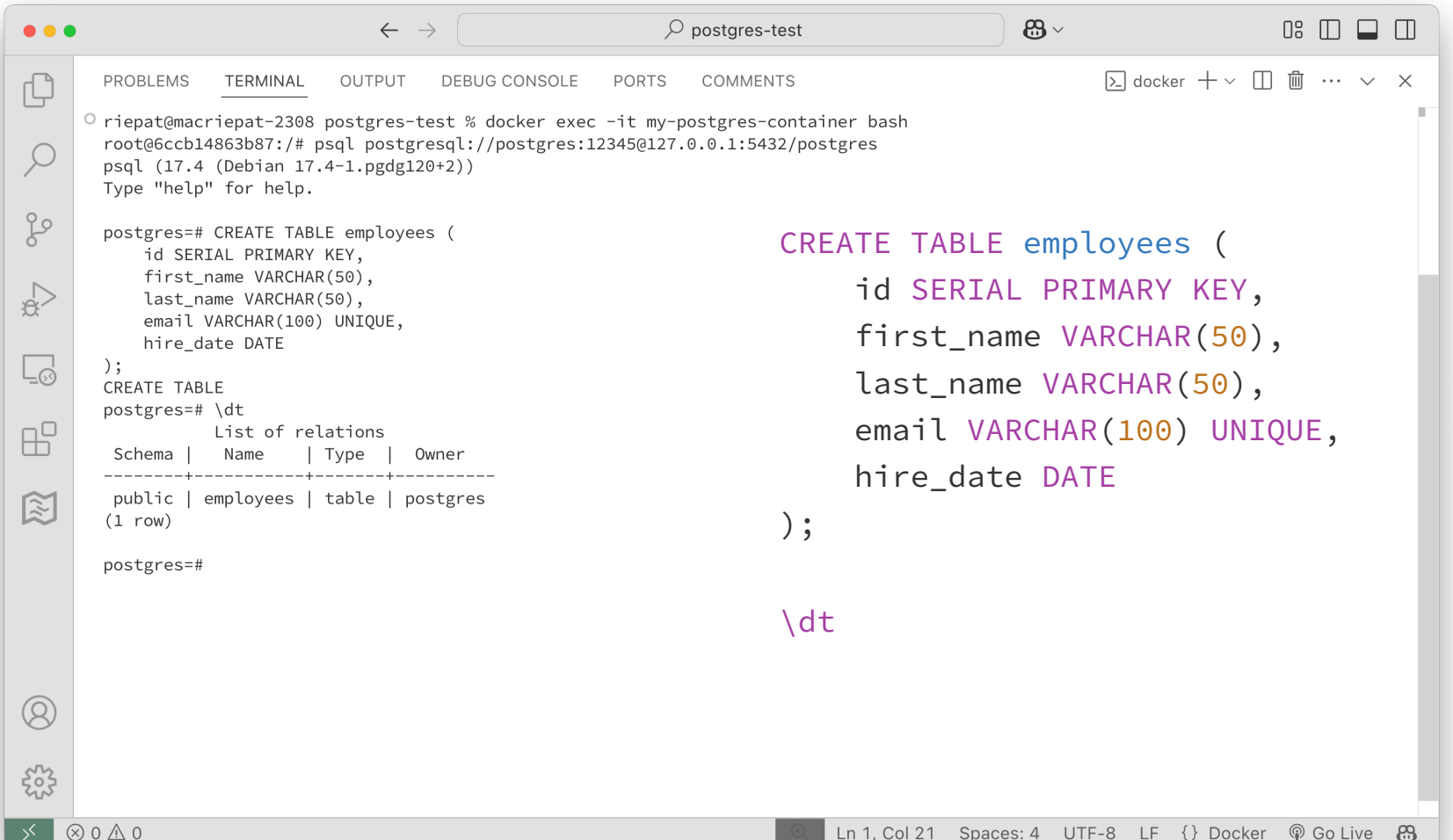
4. Delete records

```
DELETE FROM employees
WHERE id = 1;
```

```
 id | first_name | last_name |        email         | hire_date
----+------------+-----------+----------------------+------------
  2 | Patrick    | r         | patrick.r@example.com | 2023-09-01
(1 row)
```

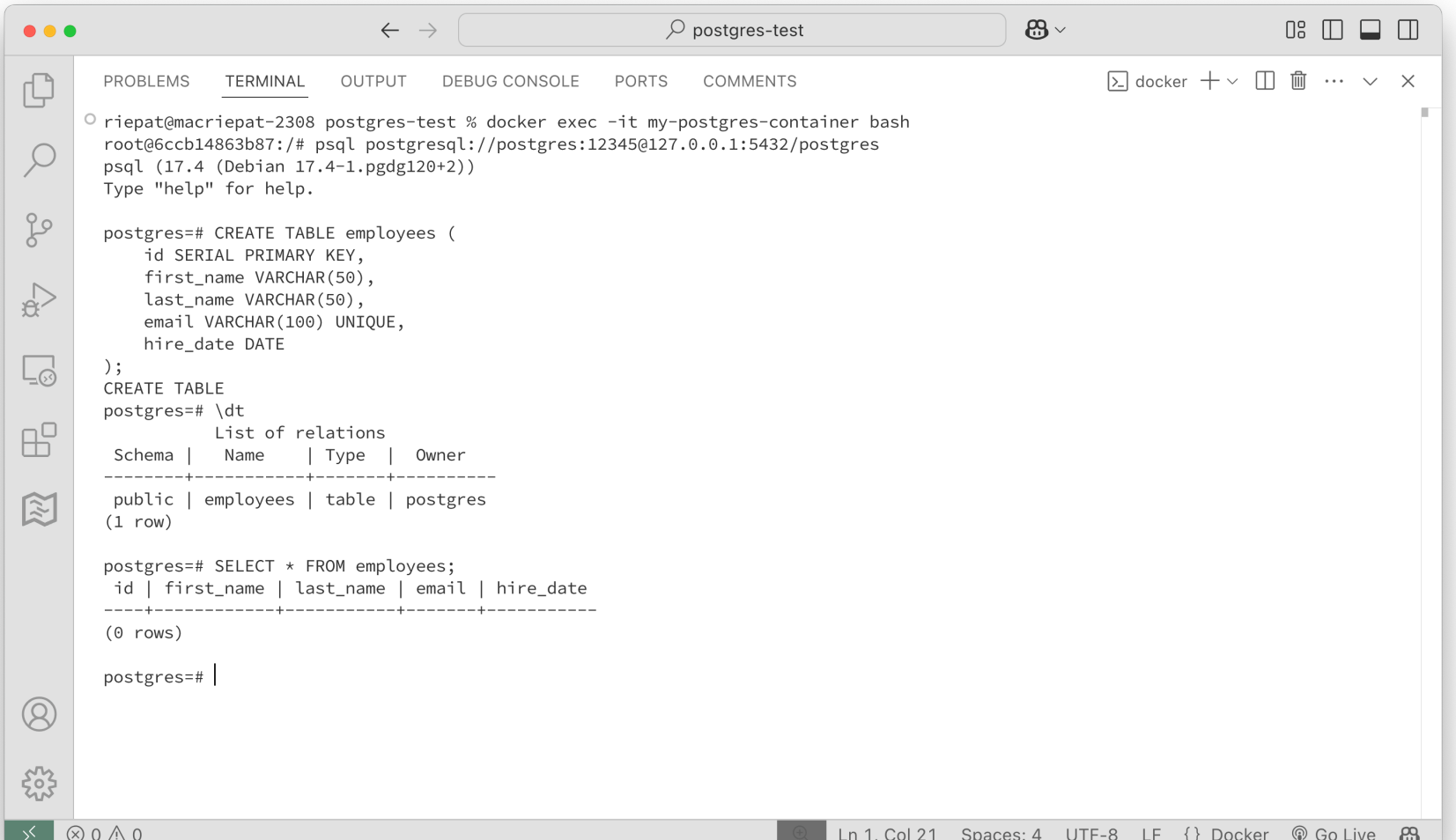
Server-Side Data Management

Postgres Database Operations



Server-Side Data Management

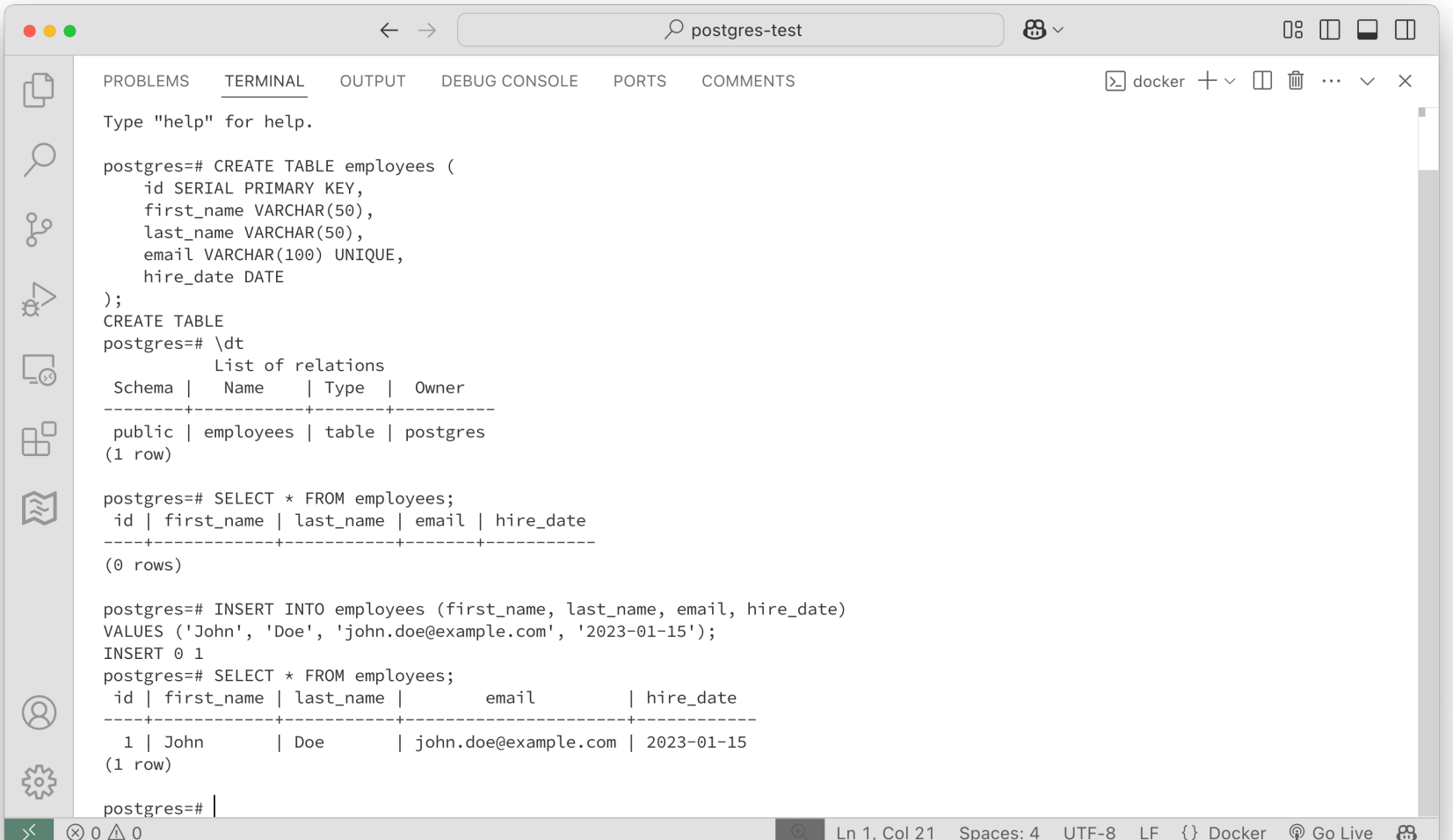
Postgres Database Operations




```
SELECT * FROM employees;
```

Server-Side Data Management

Postgres Database Operations

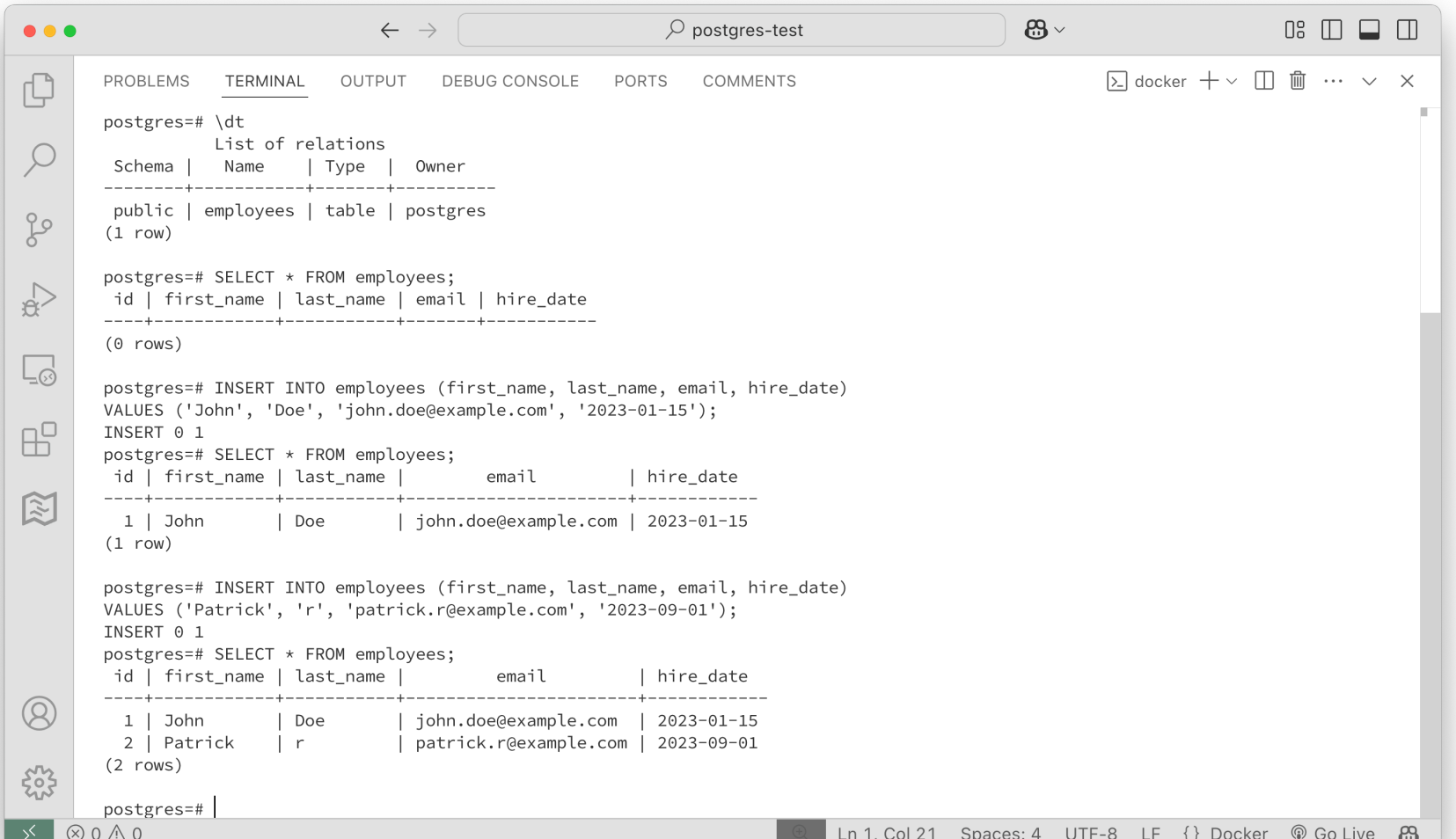



```
INSERT INTO employees (first_name, last_name, email, hire_date)
VALUES ('John', 'Doe', 'john.doe@example.com', '2023-01-15');

SELECT * FROM employees;
```

Server-Side Data Management

Postgres Database Operations

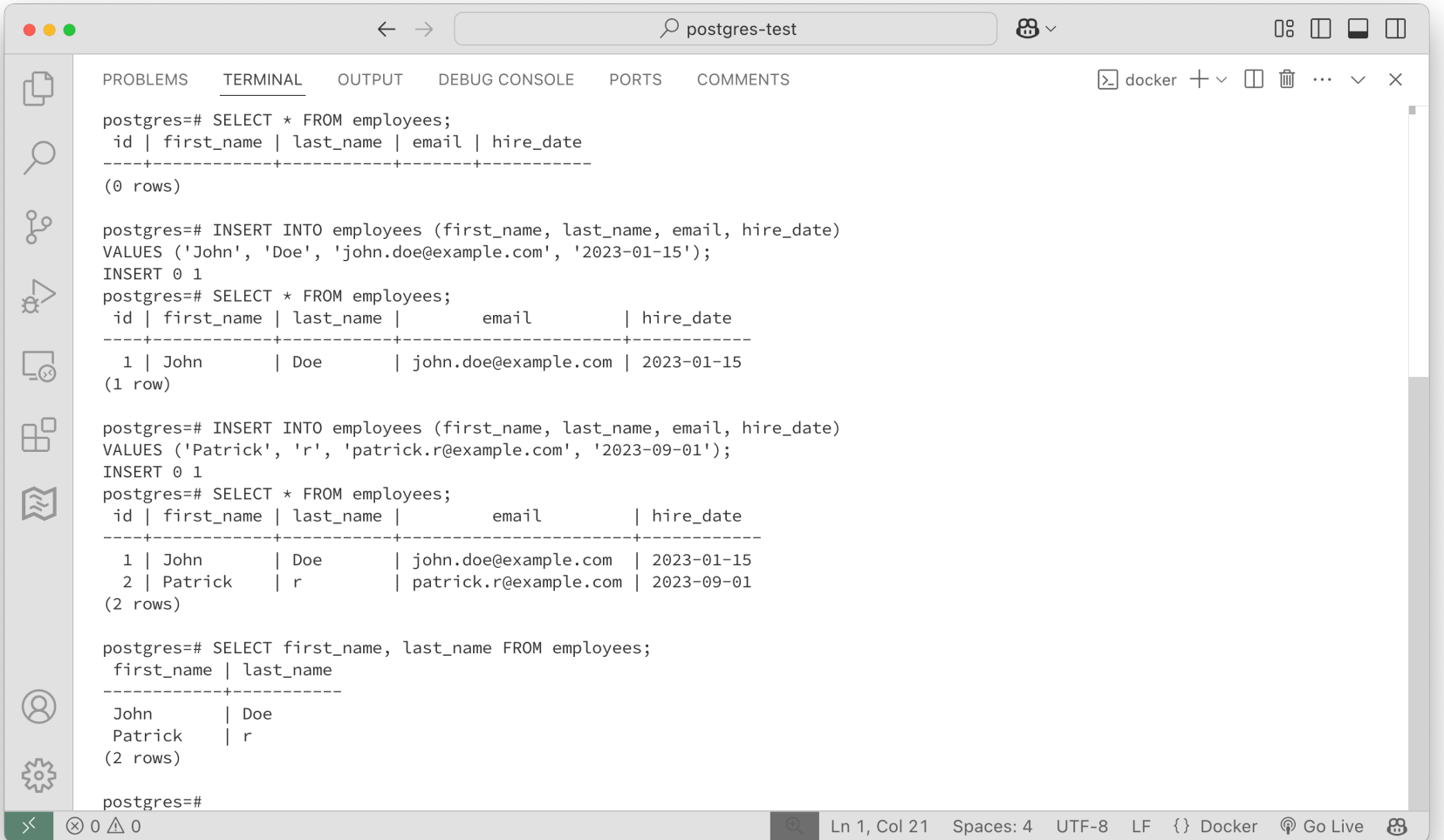



```
INSERT INTO employees (first_name, last_name, email, hire_date)
VALUES ('Patrick', 'r', 'patrick.r@example.com', '2023-09-01');

SELECT * FROM employees;
```

Server-Side Data Management

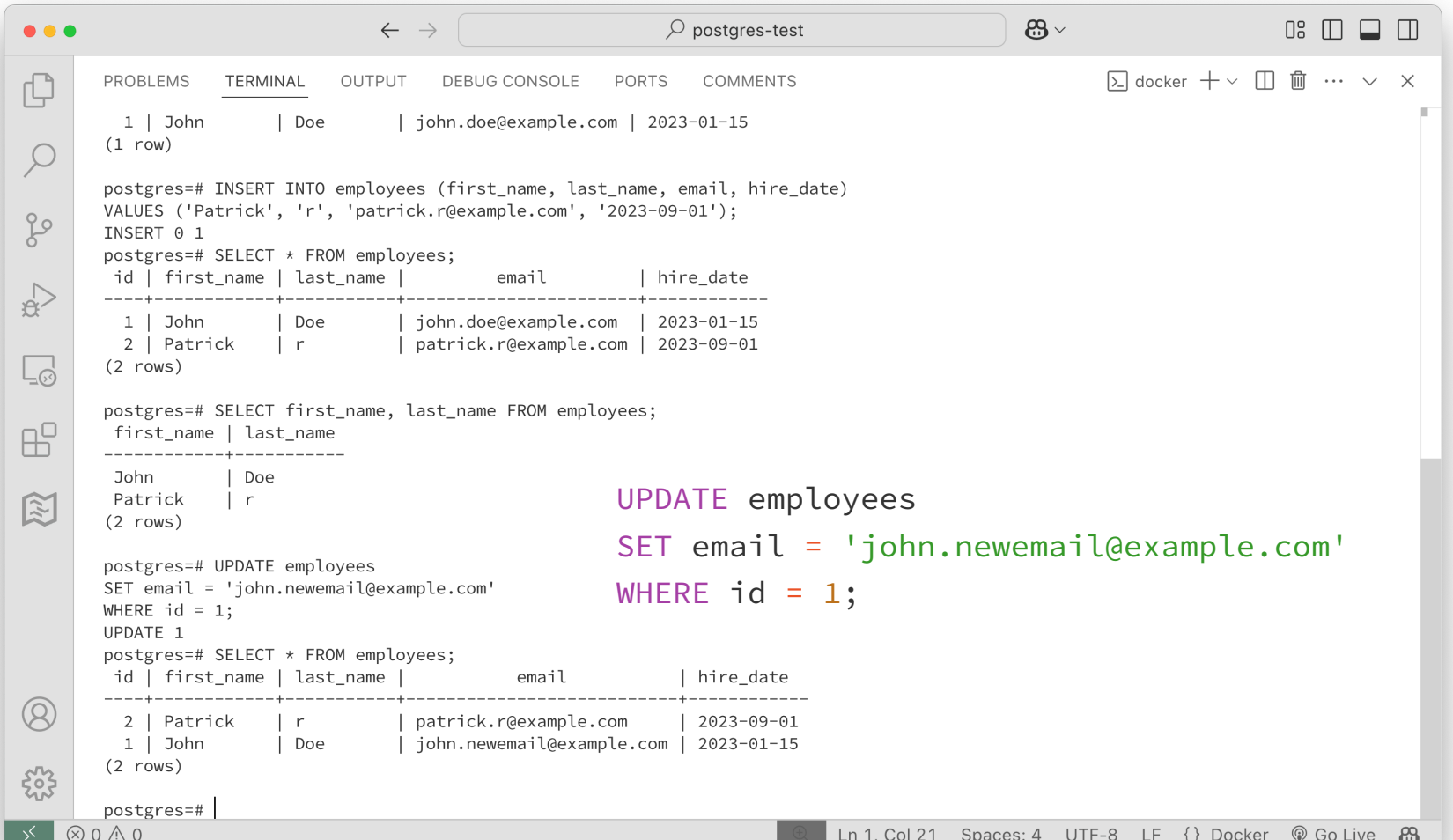
Postgres Database Operations




```
SELECT first_name, last_name FROM employees;
```

Server-Side Data Management

Postgres Database Operations

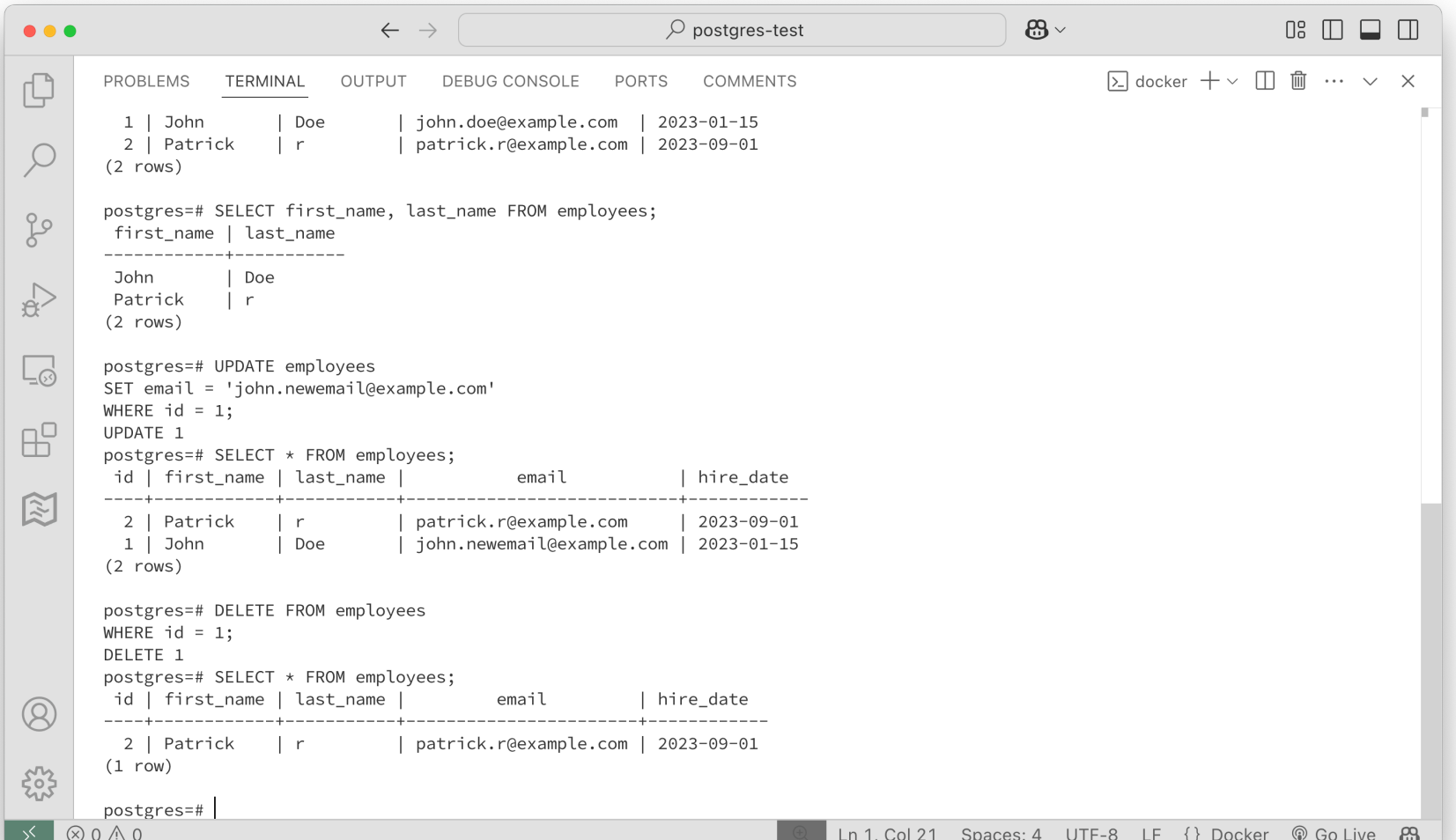



```
UPDATE employees
SET email = 'john.newemail@example.com'
WHERE id = 1;
```

```
SELECT * FROM employees;
```

Server-Side Data Management

Postgres Database Operations




```sql
DELETE FROM employees
WHERE id = 1;

SELECT * FROM employees;
```

Server-Side Data Management

Node Database Connection

1. Add `dbtest.js`,
   create a node project (`npm init`) , install `pg` (Postgres) library!

```
npm init
npm install pg
```

2. Setup a new database client in dbtest`.js` in order to connect to

```
const { Client } = require('pg');

// Configure the client to connect to your containerized PostgreSQL
const client = new Client({
  host: 'localhost',         // since the container's port is mapped to localho
  port: 5432,
  user: 'postgres',          // default user
  password: '12345',         // password set in the container command
  database: 'postgres',      // default database
});
```

3. Establish a connection to the database in the container

```
async function connectDB() {
    try {
        await client.connect();
        console.log('Connected to PostgreSQL database with async/await');
    } catch (err) {
        console.error('Connection error', err.stack);
    }
}
connectDB(); // should be called before any other function
```

4. Run dbtest locally

```
node dbtest.js
```

Server-Side Data Management

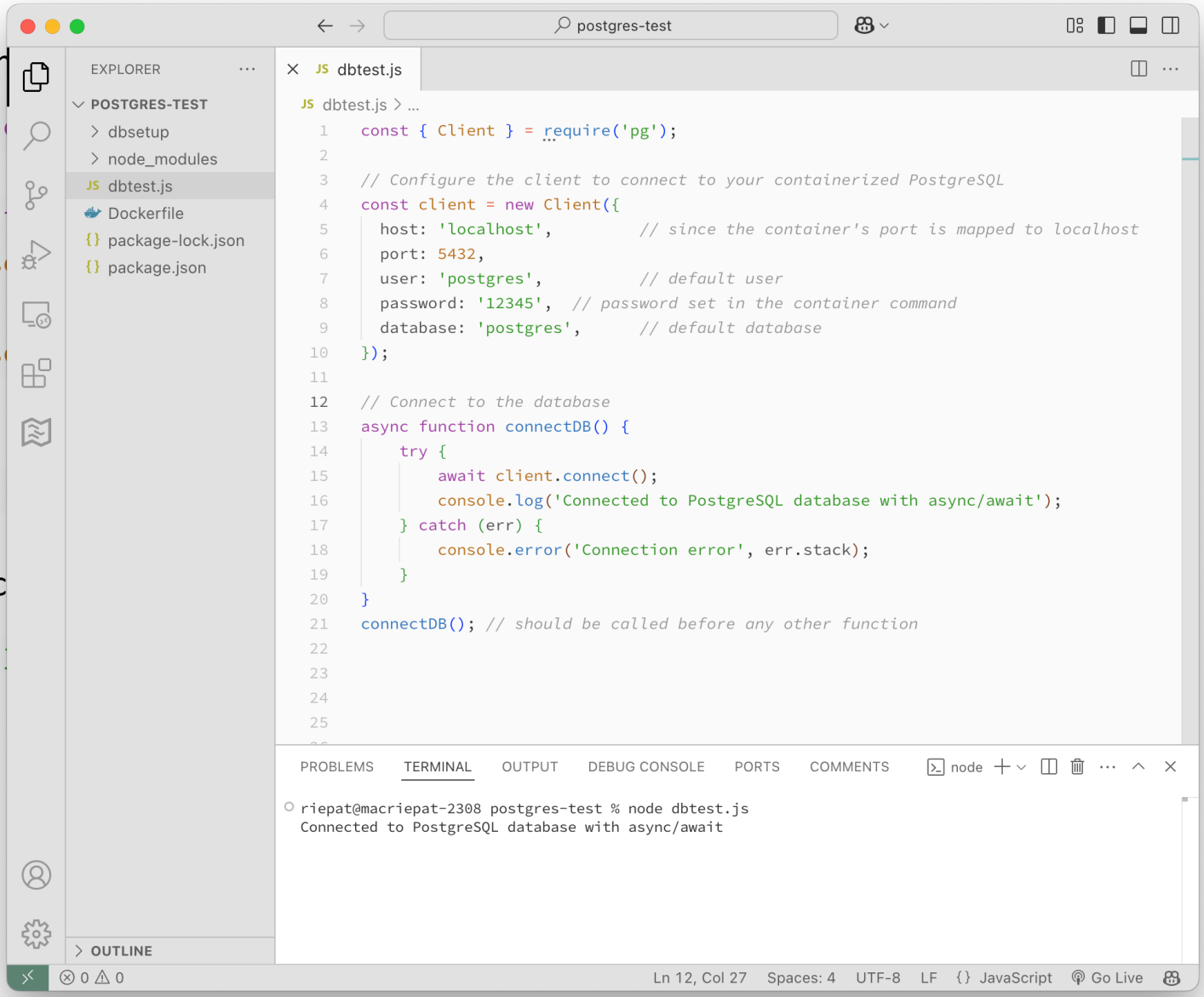
Node Database Connection

3. Establish a cor

```
async functi
    try {
        awai
        cons
    } catch
        cons
    }
}
connectDB();
```

4. Run dbtest loc

```
node dbtest.
```



Server-Side Data Management

Node Database Connection

5. Select existing records

```
async function selectRecords() {
    const selectQuery = 'SELECT * FROM employees;';
    try {
        const res = await client.query(selectQuery);
        console.log('All employees:', res.rows);
    } catch (err) {
        console.error('Error selecting records', err.stack);
    }
}
selectRecords()
```

Rerun dbtest.js

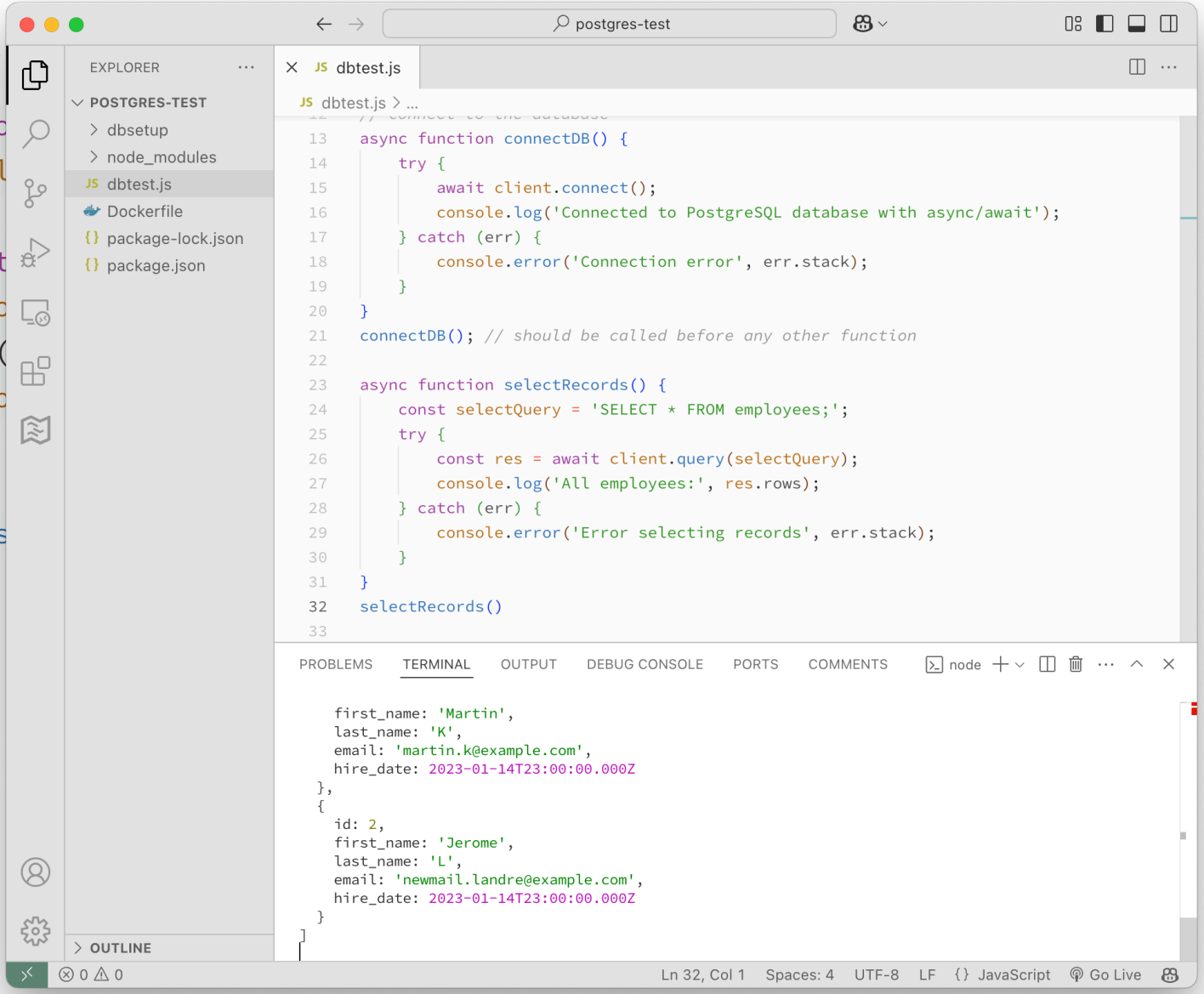
Server-Side Data Management

Node Database Connection

5. Select existing

```
async functic
    const sel
    try {
        const
        consc
    } catch (
        consc
    }
}
selectRecords
```

Rerun dbtest.js



6. Create a table in the database

```
// At first the query is simple a string created in javascript
// Multiline strings can be created with ` ` and should be closed with an ;
function createTable() {
    const createTableQuery = `
    CREATE TABLE IF NOT EXISTS employees (
      id SERIAL PRIMARY KEY,
      first_name VARCHAR(50),
      last_name VARCHAR(50),
      email VARCHAR(100) UNIQUE,
      hire_date DATE
    );      // closing SQL statement
  `;        // close string
    client.query(createTableQuery) // try await would also work with async
        .then(() => console.log('Table "employees" created or already exists'))
        .catch(err => console.error('Error creating table', err.stack));
}
createTable(); // should be called after connectDB()
```

Just EXAMPLE, for your own table!

has no effect,
if the table is already there

Rerun dbtest.js

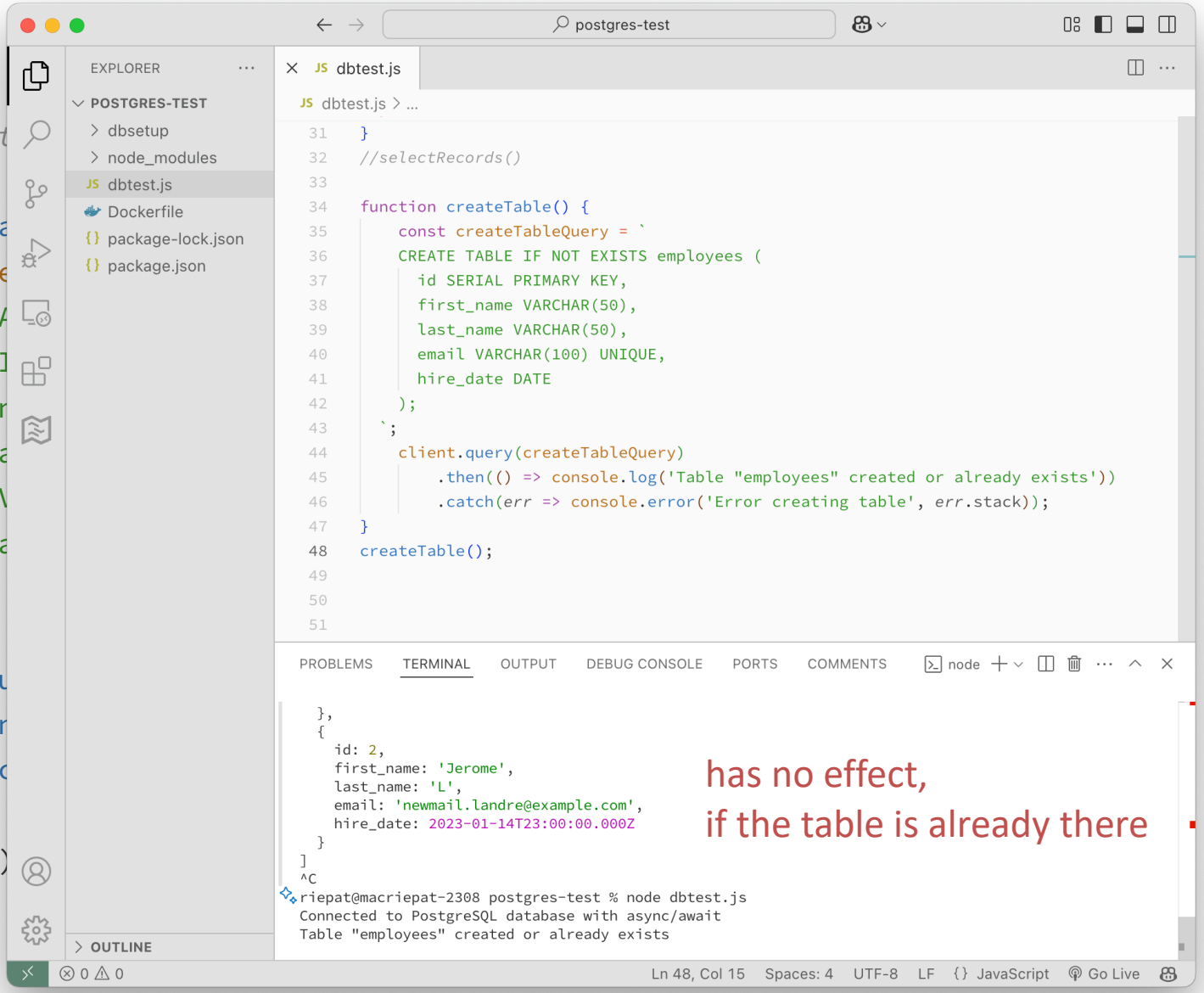
Server-Side Data Management

Node Database Connection

6. Create a table

```
// At first t
// Multiline
function crea
    const cre
    CREATE TA
      id SERI
      first_r
      last_na
      email V
      hire_da
    );
  `;
    client.qu
        .ther
        .catc
}
createTable()
```

Rerun dbtest.js



Server-Side Data Management

Node Database Connection

7. Insert data

```
function insertRecord(insertValues) {
    const insertQuery = `
    INSERT INTO employees (first_name, last_name, email, hire_date)
    VALUES ($1, $2, $3, $4)
    RETURNING *;
  `;
    client.query(insertQuery, insertValues)
        .then(res => console.log('Inserted record:', res.rows[0]))
        .catch(err => console.error('Error inserting record', err.stack));
}

const insertValues = ['You', 'YourLastame', 'you.yourlastnamel@example.com',
'2023-01-15'];
insertRecord(insertValues);
```

Placeholder for actual values

Actual values in array, sec. para

Rerun dbtest.js

Server-Side Data Management

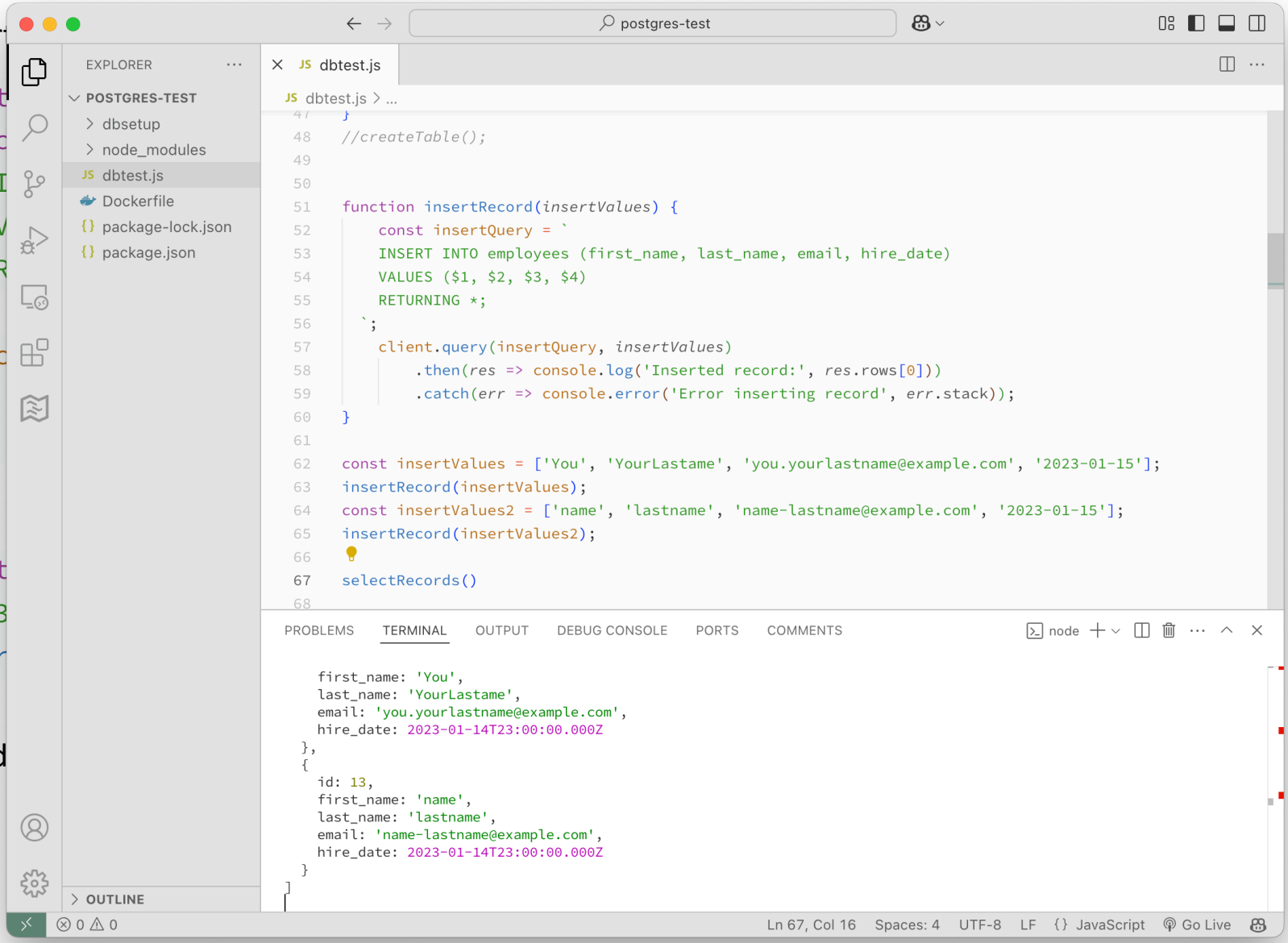
Node Database Connection

7. Inser

```
funct
    c
    I
    V
    R
`;
    c
}

const
'2023
inser
```

Rerun d



Server-Side Data Management

Node Database Connection

8. Select records (value) by value

```
async function getIDByEmail(email) {
    const selectQuery = 'SELECT id FROM employees WHERE email = $1;';
    try {
        const res = await client.query(selectQuery, [email]);
        return res.rows[0].id
    } catch (err) {
        console.error('Error getting ID', err.stack);
    }
}

getIDByEmail('patrick.r@example.com').then((id) => {
    console.log('ID:', id);
})
```

```
ID: 7
```

Rerun dbtest.js

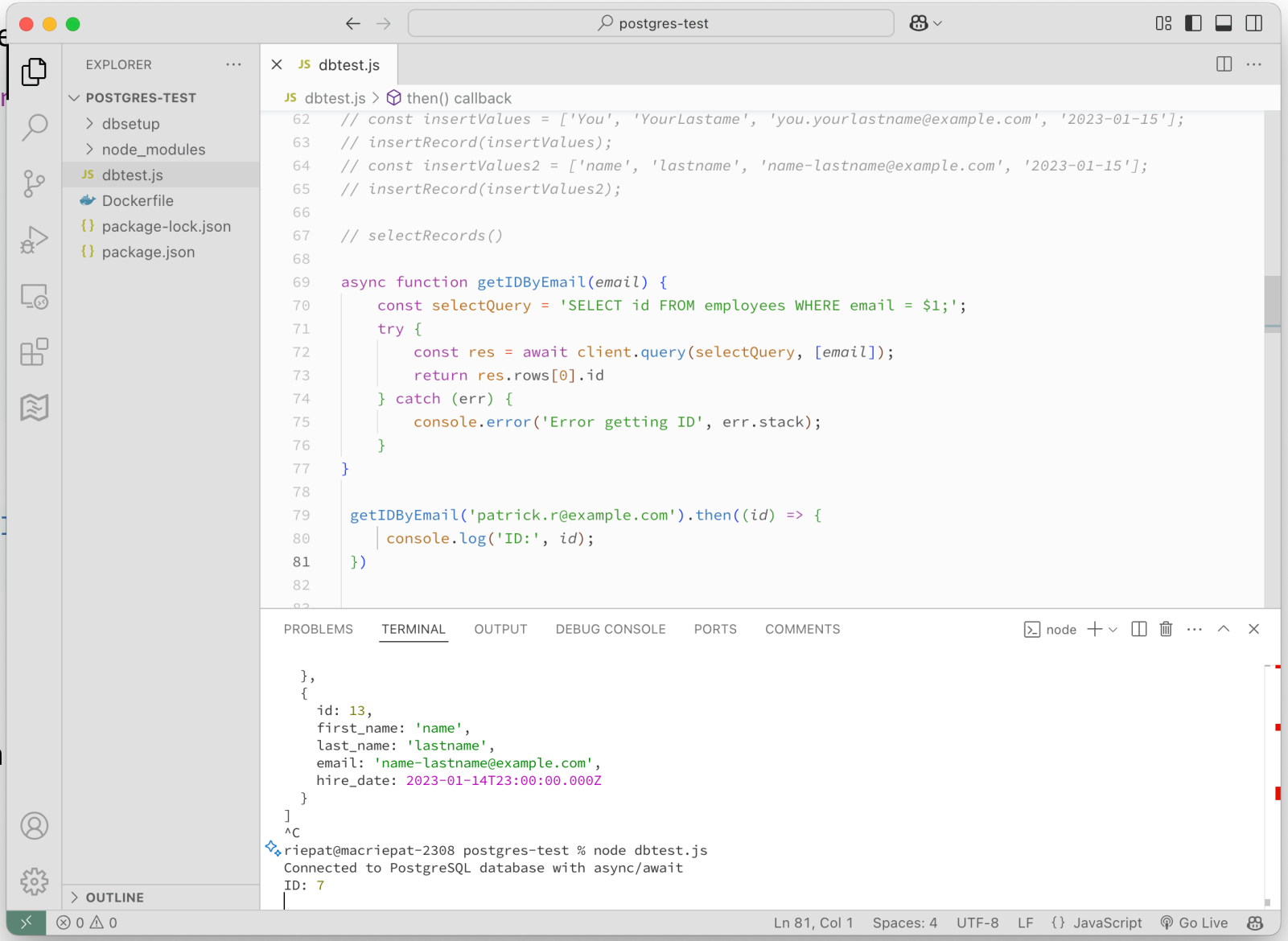
Server-Side Data Management

Node Database Connection

8. Sele

```
asyr
}
getI
})
```

Rerun



Server-Side Data Management

Node Database Connection

9. Update records

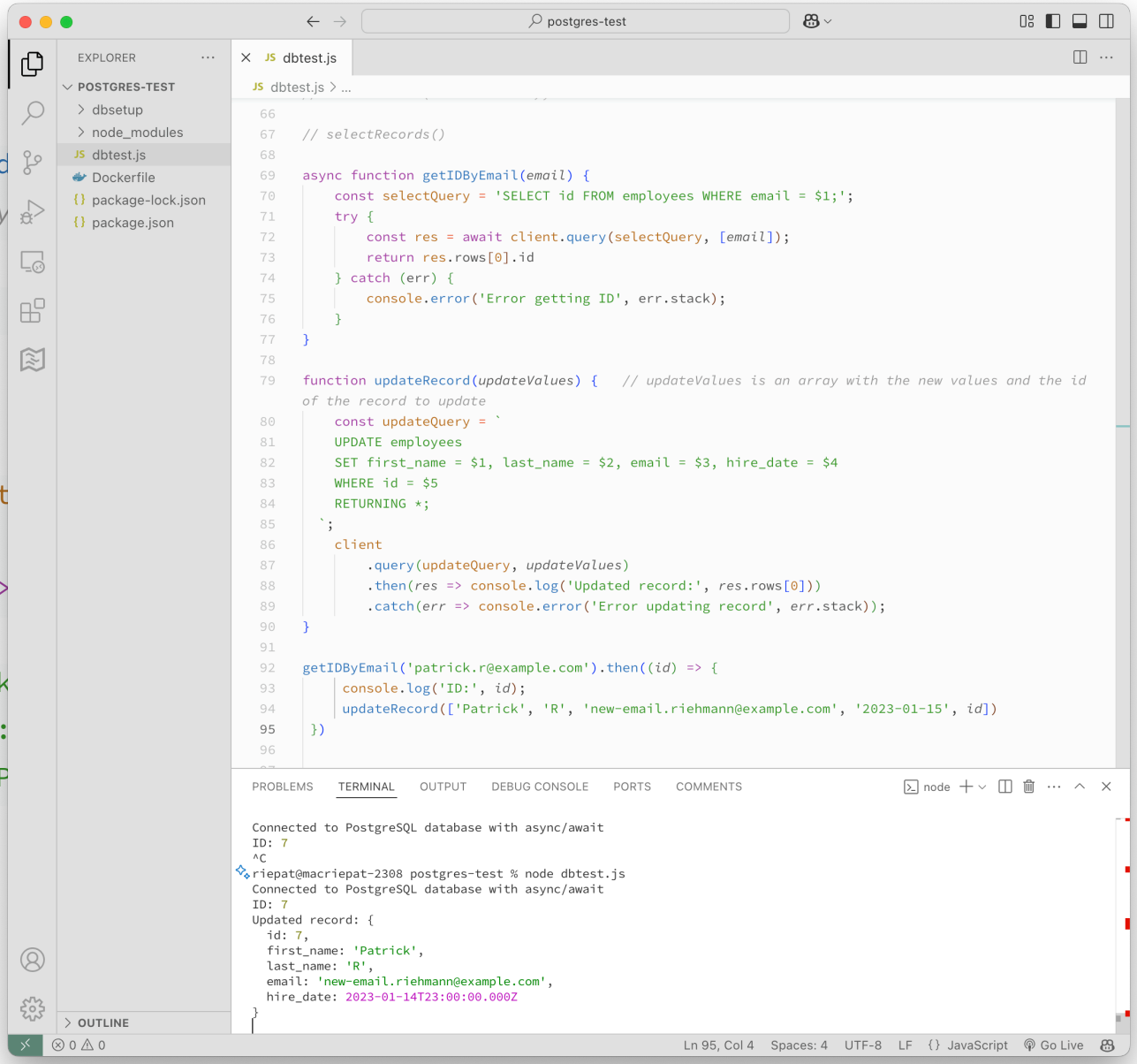
```
function updateRecord(updateValues) {
    const updateQuery = `
    UPDATE employees
    SET first_name = $1, last_name = $2, email = $3, hire_date = $4
    WHERE id = $5
    RETURNING *;
  `;
   client.query(updateQuery, updateValues)
        .then(res => console.log('Updated record:', res.rows[0]))
        .catch(err => console.error('Error updating record', err.stack));
}
getIDByEmail('patrick.r@example.com').then((id) => {
     console.log('ID:', id);
     updateRecord(['Patrick', 'R', 'new-email.riehmann@example.com', '2023-01-15', id])
 })
```

Server-Side Data Management

Node Database Connection

9. Update records

```
function updateRecord
    const updateQuery
    UPDATE employees
    SET first_name =
    WHERE id = $5
    RETURNING *;
  `;
   client.query(updat
        .then(res =>
        .catch(err =>
}
getIDByEmail('patrick
     console.log('ID:
     updateRecord(['P
 })
```



10. Delete records by id

```
function deleteRecord(id) {
// id is the primary key of the record to delete
    const deleteQuery = 'DELETE FROM employees WHERE id = $1;';
    client.query(deleteQuery, [id])
        .then(() => console.log('Record deleted with then()'))
        .catch(err => console.error('Error deleting record', err.stack));
}
deleteRecord(someID);
```

11. Close connection database

```
function disconnectDB() {
    client.end()
        .then(() => console.log('Disconnected from database with then()'))
        .catch(err => console.error('Error disconnecting', err.stack));
}
disconnectDB(); // should be called after all other functions
```

Server-Side Data Management

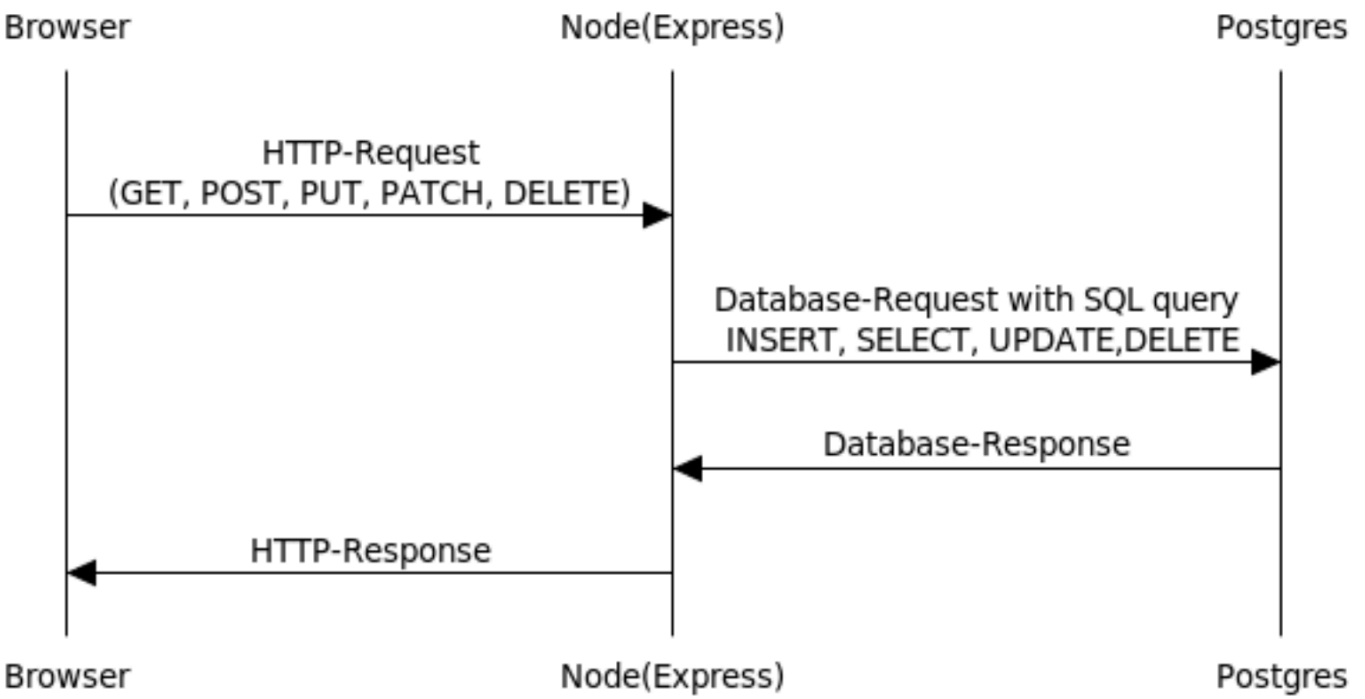
Express Database Minimal Passthrough Example - Hint Project

Challenges for the project

- ❑ Establish a database connection
- ❑ Establish the server listening on a certain port
- ❑ Creating the respective routes
- ❑ Passing through / requesting data
  - ❑ From calling a defined route to the database
  - ❑ Back from the database to the response of the called route

Server-Side Data Management

Middleware



Server-Side Data Management

Express Database Minimal Passthrough Example - Hint Project

```
const { Client } = require('pg');
const express = require('express');
const app = express();

app.get('/all', async (req, res) => {
    try {
        const dbres = await client.query('SELECT * FROM employees;');
        console.log('All employees:', dbres.rows);
        res.json(dbres.rows);
    } catch (err) {
        console.error('Error selecting records', err.stack);
    }
});

// Configure the client to connect to your containerized PostgreSQL
const client = new Client({ host: 'localhost', port: 5432,
                            user: 'postgres', password: '12345'});

// Connect to the database and start the server listening on port 3000
const startServer = async () => {
    try {
        await client.connect();
        console.log('Connected to PostgreSQL database ');
    } catch (err) {
        console.error('Connection error', err.stack);
    }
    app.listen(3000, () => {
      console.log('Example app listening on port 3000!');
    });
}
startServer();
```

Server-Side Data Management

Express Database Minimal Passthrough Example - Hint Project

```
const { Client } = require('pg');
const express = require('express');
const app = express();

app.get('/all', async (req, res) => {
    try {
        const dbres = await client.query('SELECT * FROM employees;');
        console.log('All employees:', dbres.rows);
        res.json(dbres.rows);
    } catch (err) {
        console.error('Error selecting records', err.stack);
    }
});

// Configure the client to connect to your containerized PostgreSQL
const client = new Client({ host: 'localhost', port: 5432,
                            user: 'postgres', password: '12345'});

// Connect to the database and start the server listening on port 3000
const startServer = async () => {
    try {
        await client.connect();
        console.log('Connected to PostgreSQL database ');
    } catch (err) {
        console.error('Connection error', err.stack);
    }
    app.listen(3000, () => {
      console.log('Example app listening on port 3000!');
    });
}
startServer();
```

Route definition

Database transaction

Response sending

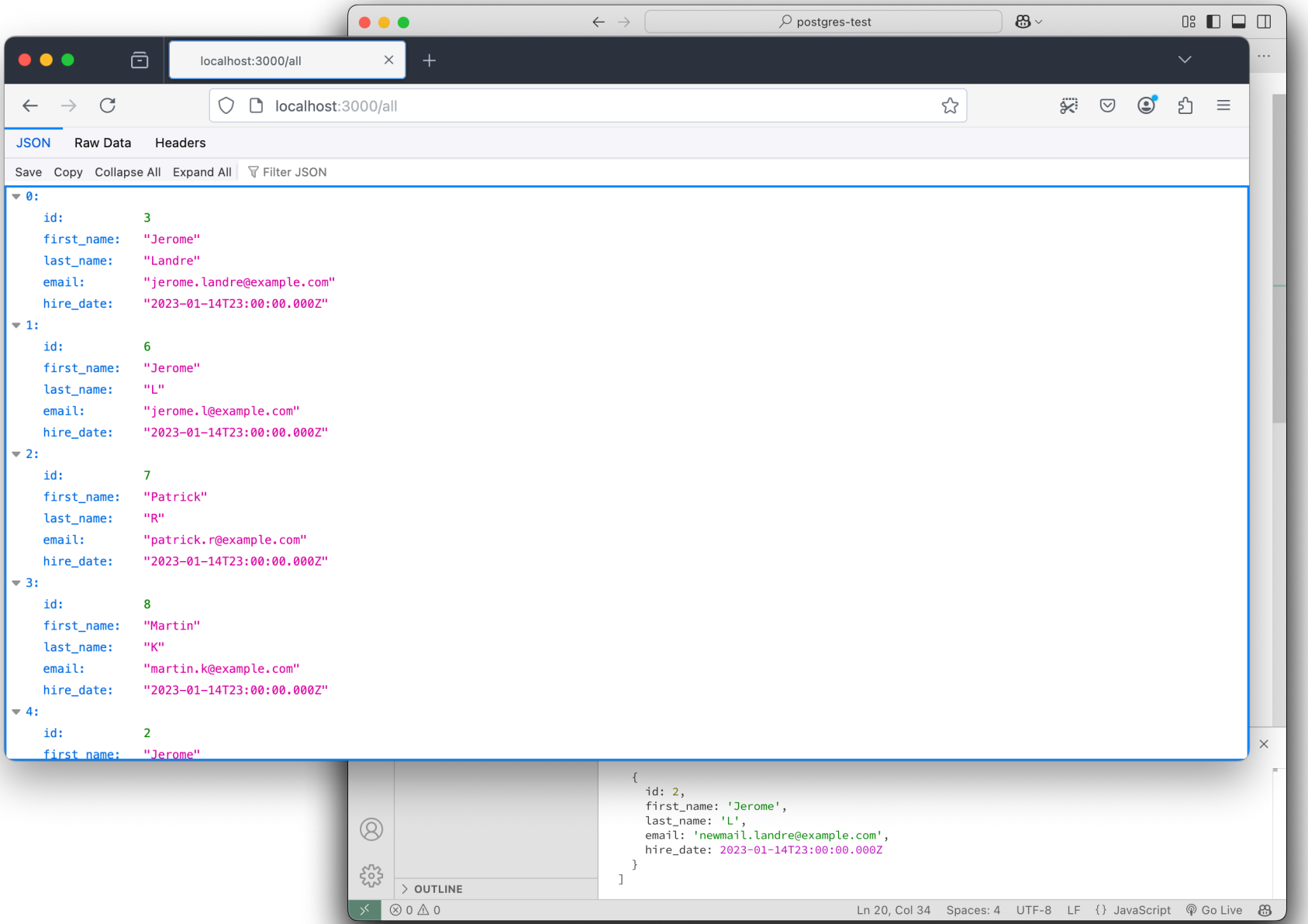
Database config

Database connection

Server listen on port

Server-Side Data Management

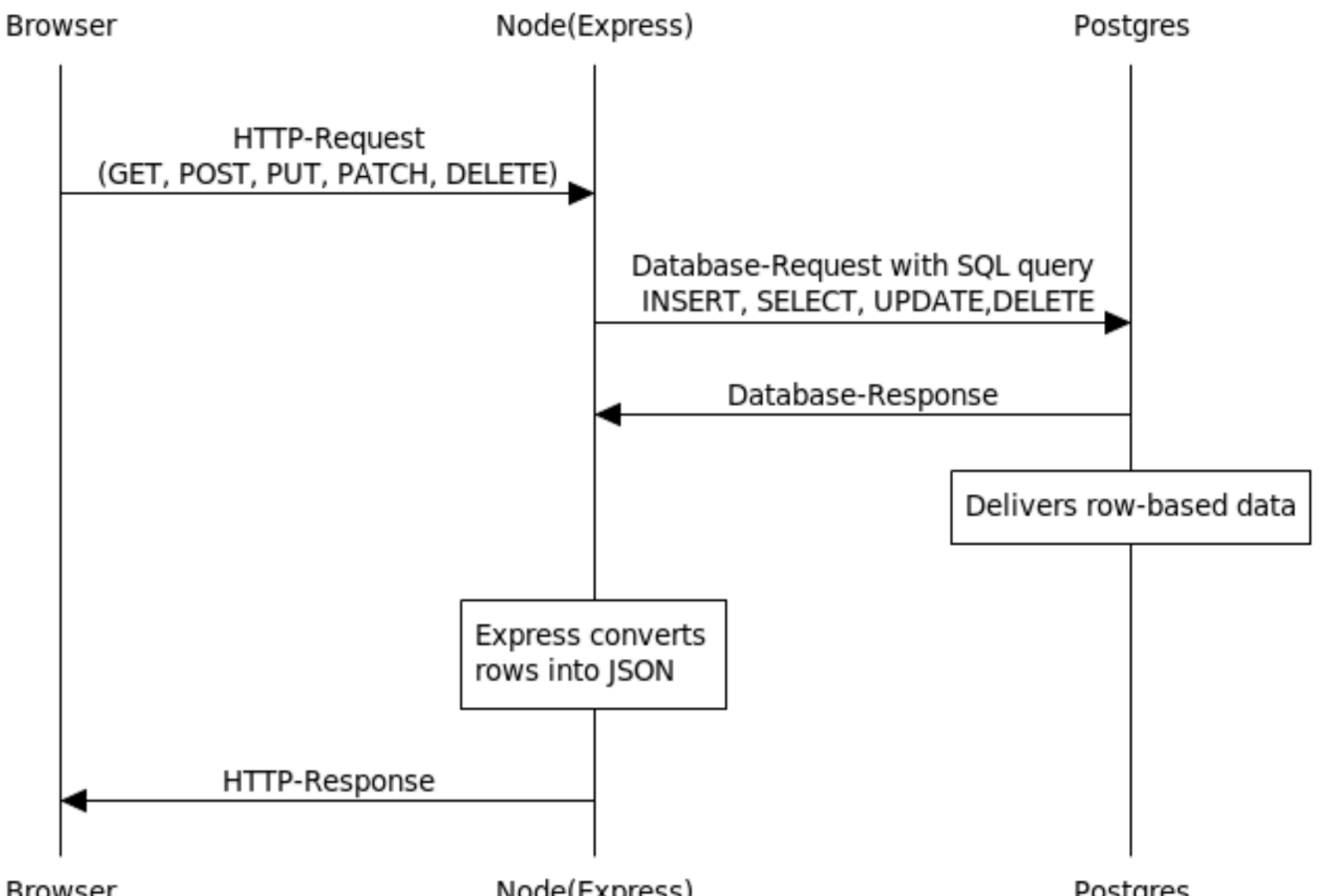
Express Database Minimal Passthrough Example



Server-Side Data Management

Express Database Minimal Passthrough Example - Hint Project

Middleware - also known as "architectural glue" - implements the infrastructure for and between components.



Problems with the minimal example

- Not really good in terms of software design
- Better approach would be to create a `Model` class that wraps and handles the connection to the database
- Do some research how to split up the minimal example into a `Model` class in a file `model.js` and the file `server.js` with the express server

Server-Side Data Management

Middleware

Better Architecture: Can be called Model-Controller