

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
```

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
```

*Just Overview,
we go along step-by-step*

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

The screenshot shows the Visual Studio Code (VS Code) interface with the following details:

- Explorer View:** Shows a folder named "POSTGRES-TEST" containing a "Dockerfile".
- Dockerfile Content:**

```
1 FROM postgres:latest
2 ENV POSTGRES_USER postgres
3 ENV POSTGRES_PASSWORD 12345
4 ENV POSTGRES_DB postgres
5
6 EXPOSE 5432
7
```
- Terminal View:** Displays the command: `riePAT@macriepat-2308 postgres-test % docker build -t my-postgres-image .`
- Status Bar:** Shows icons for file status (0 errors, 0 warnings), search, file navigation, line 1, column 1, spaces: 4, UTF-8, LF, Docker, Go Live, and a user icon.

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

Server-Side Data Management

Postgres Database Operations

The screenshot shows a code editor interface with a Dockerfile open. The Dockerfile defines a PostgreSQL container with the following configuration:

```
FROM postgres:latest
ENV POSTGRES_USER postgres
ENV POSTGRES_PASSWORD 12345
ENV POSTGRES_DB postgres
EXPOSE 5432
```

The editor has tabs for PROBLEMS, TERMINAL, OUTPUT, DEBUG CONSOLE, PORTS, and COMMENTS. The PROBLEMS tab shows a warning about naming the image to docker.io/library/my-postgres-image. It also lists four warnings related to legacy key-value format and secrets usage.

The TERMINAL tab shows the command used to run the Docker container:

```
riePAT@macbook-Pro:~/Documents/Docker$ docker run -d -p 5432:5432 --name my-postgres-container my-postgres-image
```

The OUTPUT tab shows the container's logs, including its creation time and status:

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
6ccb14863b87        my-postgres-image   "docker-entrypoint.s..."   32 seconds ago     Up 31 seconds      0.0.0.0:5432->5432/tcp   my-postgres-
```

The bottom status bar indicates the current terminal session and various system metrics like CPU and memory usage.

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

Server-Side Data Management

Postgres Database Operations

The screenshot shows a VS Code interface with a Dockerfile open in the editor. The Dockerfile defines a PostgreSQL container with environment variables and port 5432 exposed. A build message indicates the image was named to docker.io/library/my-postgres-image. The terminal shows a Docker run command and a ps command listing the running container. A blue arrow points to the ps command in the terminal.

```
Dockerfile
FROM postgres:latest
ENV POSTGRES_USER postgres
ENV POSTGRES_PASSWORD 12345
ENV POSTGRES_DB postgres
EXPOSE 5432
```

PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE PORTS COMMENTS

=> => naming to docker.io/library/my-postgres-image 0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/p7ec78syl4zyjzpqlmx0k2os2w

4 warnings found (use docker --debug to expand):

- LegacyKeyValueFormat: "ENV key=value" should be used instead of legacy "ENV key value" format (line 4)
- LegacyKeyValueFormat: "ENV key=value" should be used instead of legacy "ENV key value" format (line 2)
- LegacyKeyValueFormat: "ENV key=value" should be used instead of legacy "ENV key value" format (line 3)
- SecretsUsedInArgOrEnv: Do not use ARG or ENV instructions for sensitive data (ENV "POSTGRES_PASSWORD") (line 3)

What's next:

View a summary of image vulnerabilities and recommendations → [docker scout quickview](#)

- riepat@macriepat-2308 postgres-test % docker run -d -p 5432:5432 --name my-postgres-container my-postgres-image 6ccb14863b8703503609844f88969298af0a17215a421a47c0f71e2658288adf
- riepat@macriepat-2308 postgres-test % docker ps

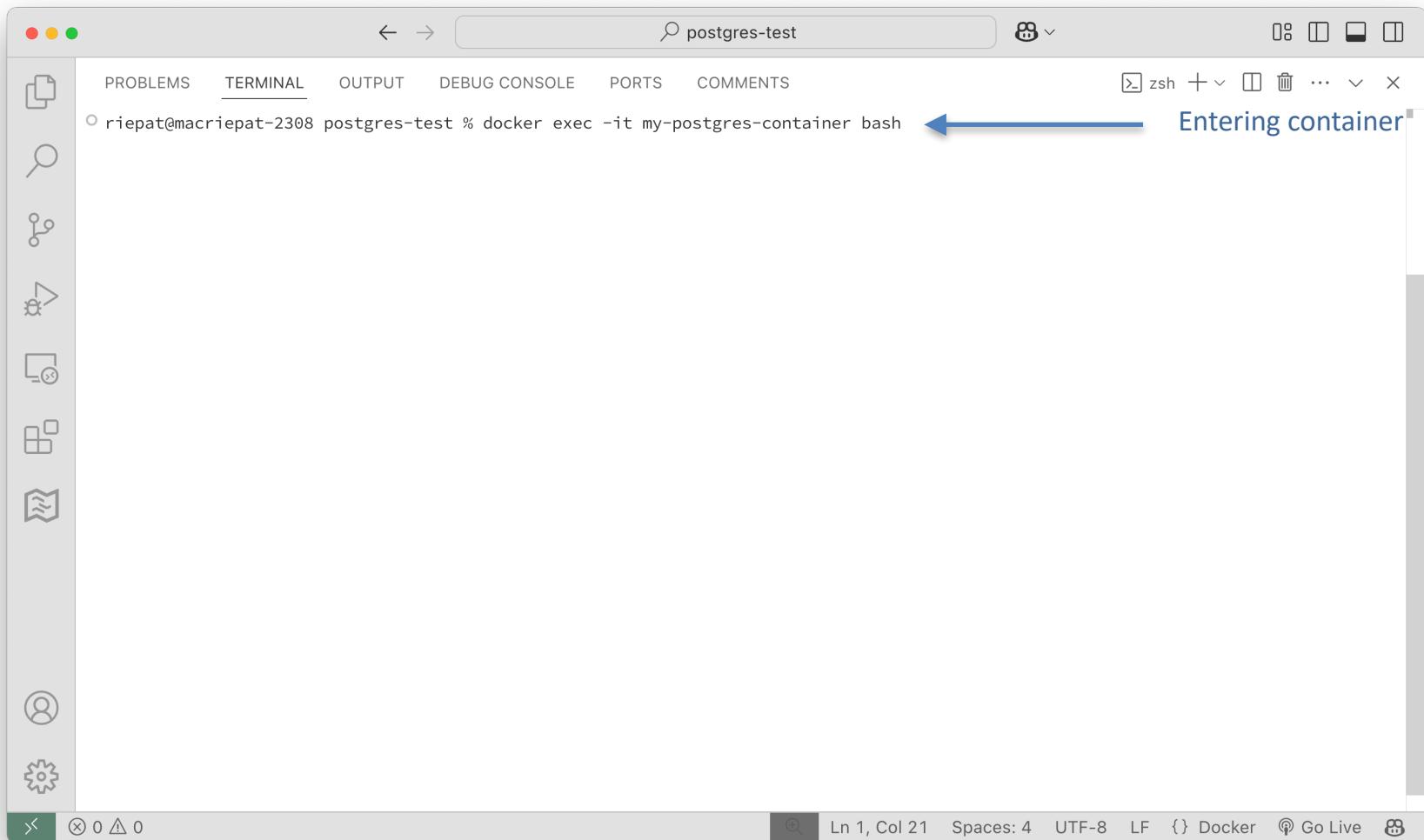
COLUMN	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1	6ccb14863b87	my-postgres-image	"docker-entrypoint.s..."	32 seconds ago	Up 31 seconds	0.0.0.0:5432->5432/tcp	my-postgres-
2	riepat@macriepat-2308	postgres-test	%				

Ln 1, Col 21 Spaces: 4 UTF-8 LF {} Docker Go Live

docker ps

Server-Side Data Management

Postgres Database Operations



A screenshot of the Visual Studio Code (VS Code) interface, specifically the Terminal tab, showing a session in a Docker container. The terminal window title is "postgres-test". The status bar at the bottom shows the command: "docker exec -it my-postgres-container bash". A blue arrow points from the text "Entering container" to the command in the status bar.

The terminal output shows:

```
riePAT@macbook-pro-2308 postgres-test % docker exec -it my-postgres-container bash
```

The status bar at the bottom of the terminal window displays the command: "docker exec -it my-postgres-container bash".

docker exec -it my-postgres-container bash

Server-Side Data Management

Postgres Database Operations

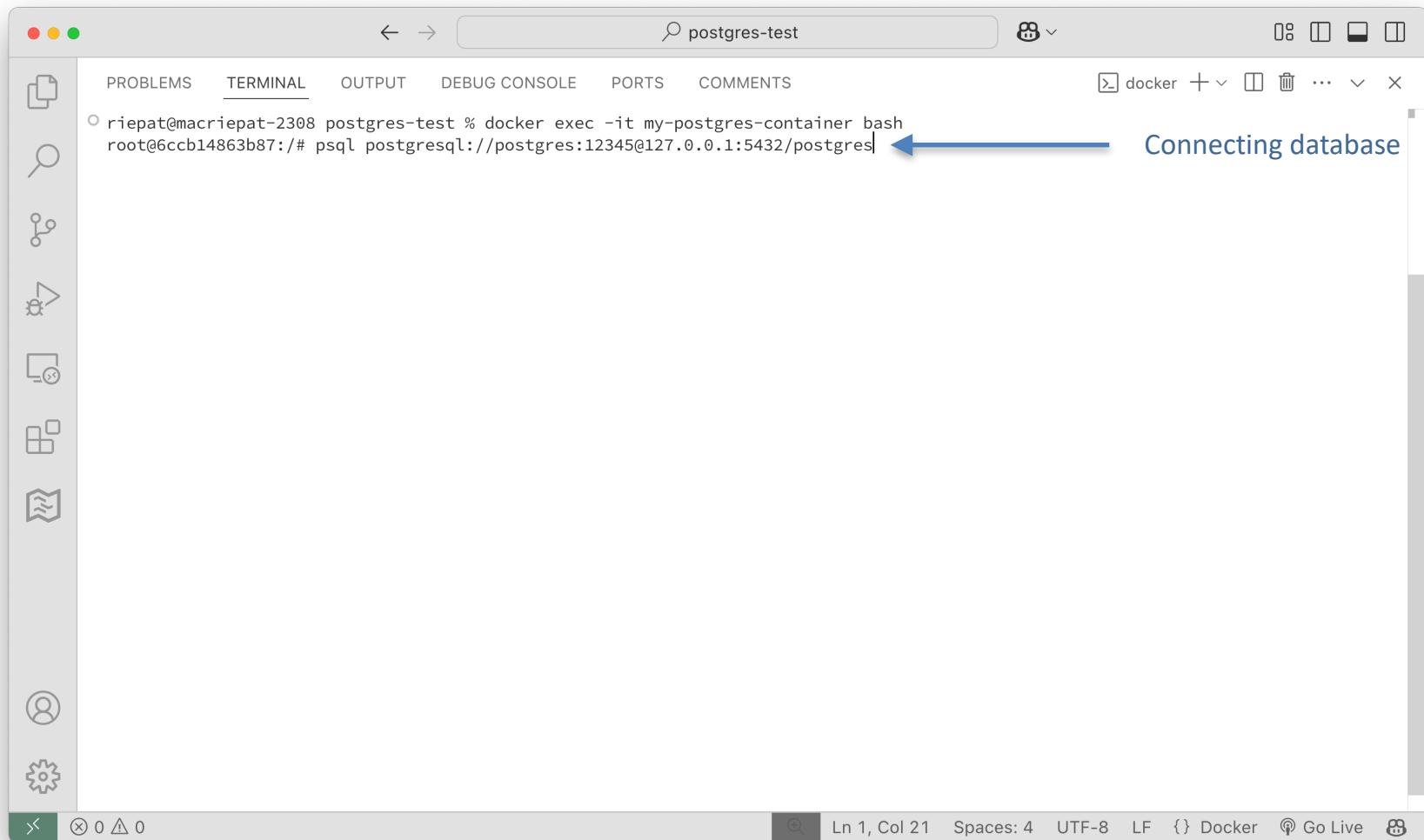
The screenshot shows a terminal window within a code editor interface. The title bar reads "postgres-test". The top menu bar includes standard icons for file operations and a "docker" tab. The left sidebar contains various icons for navigation and management. The main terminal area displays the following command and its output:

```
riePAT@macriepat-2308 postgres-test % docker exec -it my-postgres-container bash
root@6ccb14863b87:/# |
```

The status bar at the bottom shows "Ln 1, Col 21" and other terminal metadata.

Server-Side Data Management

Postgres Database Operations



A screenshot of a terminal window within a code editor interface. The title bar says "postgres-test". The menu bar includes "PROBLEMS", "TERMINAL" (which is underlined), "OUTPUT", "DEBUG CONSOLE", "PORTS", and "COMMENTS". On the right side of the terminal, there are icons for "docker", a "+" sign, a trash can, and other standard file operations. A blue arrow points from the text "Connecting database" to the command being typed in the terminal.

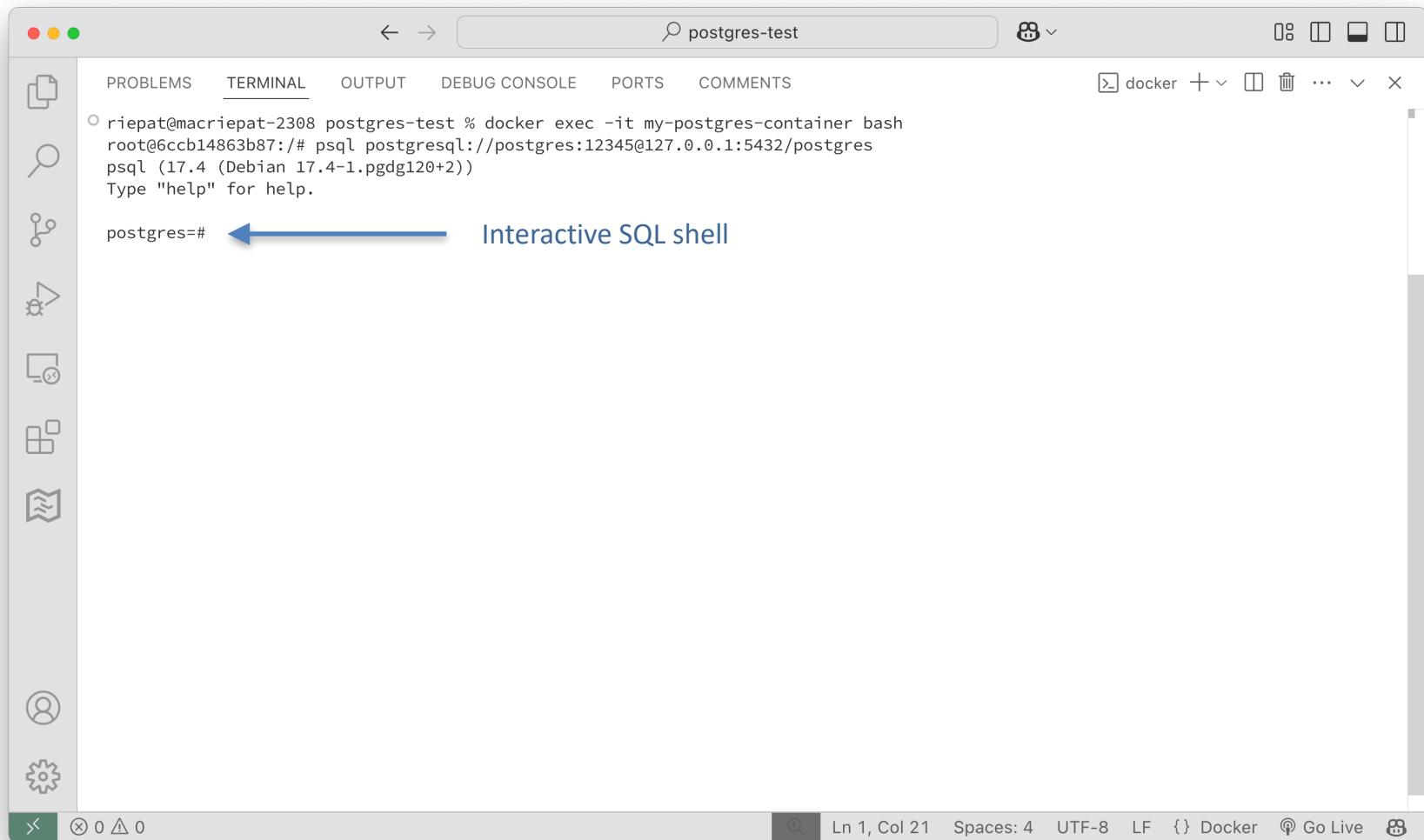
```
riePAT@macriepat-2308 postgres-test % docker exec -it my-postgres-container bash
root@6ccb14863b87:/# psql postgresql://postgres:12345@127.0.0.1:5432/postgres|
```

The status bar at the bottom shows "Ln 1, Col 21" and "Spaces: 4". There are also icons for "Docker", "Go Live", and a "git" icon.

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

Server-Side Data Management

Postgres Database Operations



A screenshot of a terminal window in the Visual Studio Code (VS Code) interface. The title bar shows the workspace name "postgres-test". The left sidebar contains icons for various extensions or features. The main area is a terminal window with the following content:

```
riePAT@macriepat-2308 postgres-test % docker exec -it my-postgres-container bash
root@6ccb14863b87:/# psql postgresql://postgres:12345@127.0.0.1:5432/postgres
psql (17.4 (Debian 17.4-1.pgdg120+2))
Type "help" for help.

postgres=#
```

An arrow points from the text "Interactive SQL shell" to the prompt "postgres=#". The status bar at the bottom shows the current file path, line number (Ln 1, Col 21), and other terminal settings.

Server-Side Data Management

Postgres Database Operations

The screenshot shows a terminal session in VS Code connected to a PostgreSQL container named 'postgres-test'. The session starts with the user entering a Docker command to exec into the container, followed by a psql command to connect to the database. Once connected, the user is at the PostgreSQL prompt, 'postgres=#'.

Annotations on the right side of the terminal window explain the steps:

- A blue double-headed arrow points from the 'docker' icon in the top right to the 'docker' command in the terminal, labeled "Entering container".
- A blue double-headed arrow points from the 'psql' command in the terminal to the database connection string, labeled "Connecting database".
- A blue double-headed arrow points from the PostgreSQL prompt 'postgres=' to the text 'Interactive SQL shell', labeled "Interactive SQL shell".

```
riePAT@macriepat-2308 postgres-test % docker exec -it my-postgres-container bash
root@6ccb14863b87:/# psql postgresql://postgres:12345@127.0.0.1:5432/postgres
psql (17.4 (Debian 17.4-1.pgdg120+2))
Type "help" for help.

postgres=#
```

VS Code interface elements visible include the title bar 'postgres-test', the terminal tab, and the Docker extension's sidebar icon.

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');
```

*We go along step-by-step
Just Overview,*

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)				

We go along overview, just step-by-step

Server-Side Data Management

Postgres Database Operations

A screenshot of a terminal window titled "postgres-test". The window includes a navigation bar with icons for PROBLEMS, TERMINAL, OUTPUT, DEBUG CONSOLE, PORTS, and COMMENTS. On the right, there are icons for docker, a plus sign, a minus sign, a trash can, and a close button. The terminal content shows a session in a PostgreSQL container:

```
riePAT@macriepat-2308 postgres-test % docker exec -it my-postgres-container bash
root@6ccb14863b87:/# psql postgresql://postgres:12345@127.0.0.1:5432/postgres
psql (17.4 (Debian 17.4-1.pgdg120+2))
Type "help" for help.

postgres=# CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    hire_date DATE
);
CREATE TABLE
postgres=# \dt
              List of relations
 Schema |   Name    | Type  | Owner
-----+-----------+-----+-----
 public | employees | table | postgres
(1 row)

postgres=#

```

The right side of the image displays the SQL code for creating the "employees" table and listing relations:

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

\dt
```

Server-Side Data Management

Postgres Database Operations

A screenshot of a terminal window titled "postgres-test". The window includes a sidebar with various icons for file management, search, and system status. The main area displays a PostgreSQL session. The session starts with a connection command, followed by the creation of a "employees" table with columns for id, first_name, last_name, email, and hire_date. A \dt command shows the table in the public schema. A SELECT * FROM employees query returns zero rows. The bottom of the window shows the command "SELECT * FROM employees;" entered in the terminal.

```
riePAT@macriePAT-2308 postgres-test % docker exec -it my-postgres-container bash
root@6ccb14863b87:/# psql postgresql://postgres:12345@127.0.0.1:5432/postgres
psql (17.4 (Debian 17.4-1.pgdg120+2))
Type "help" for help.

postgres=# CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    hire_date DATE
);
CREATE TABLE
postgres=# \dt
List of relations
 Schema |   Name    | Type  | Owner
-----+-----+-----+-----
 public | employees | table | postgres
(1 row)

postgres=# SELECT * FROM employees;
 id | first_name | last_name | email | hire_date
----+-----+-----+-----+
(0 rows)

postgres=#

```

```
SELECT * FROM employees;
```

Server-Side Data Management

Postgres Database Operations

The screenshot shows a PostgreSQL terminal window titled "postgres-test". The terminal interface includes a header with tabs for PROBLEMS, TERMINAL (which is selected), OUTPUT, DEBUG CONSOLE, PORTS, and COMMENTS. A dock bar at the top right contains icons for Docker, a plus sign, a trash can, and other controls. The main area displays the following PostgreSQL session:

```
postgres=# CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    hire_date DATE
);
CREATE TABLE
postgres=# \dt
      List of relations
 Schema |   Name    | Type  | Owner
-----+-----+-----+
 public | employees | table | postgres
(1 row)

postgres=# SELECT * FROM employees;
 id | first_name | last_name | email | hire_date
----+-----+-----+-----+
(0 rows)

postgres=# INSERT INTO employees (first_name, last_name, email, hire_date)
VALUES ('John', 'Doe', 'john.doe@example.com', '2023-01-15');
INSERT 0 1
postgres=# SELECT * FROM employees;
 id | first_name | last_name |      email       | hire_date
----+-----+-----+-----+-----+
  1 | John        | Doe        | john.doe@example.com | 2023-01-15
(1 row)

postgres=|
```

The bottom status bar shows the cursor position (Ln 1, Col 21), text width (Spaces: 4), encoding (UTF-8), line feed (LF), and Docker integration status.

```
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

The screenshot shows a PostgreSQL terminal window titled "postgres-test". The terminal displays the following commands and their results:

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

postgres=# SELECT * FROM employees;
 id | first_name | last_name | email | hire_date
-----+-----+-----+-----+
(0 rows)

postgres=# INSERT INTO employees (first_name, last_name, email, hire_date)
VALUES ('John', 'Doe', 'john.doe@example.com', '2023-01-15');
INSERT 0 1
postgres=# SELECT * FROM employees;
 id | first_name | last_name |      email       | hire_date
-----+-----+-----+-----+
 1 | John       | Doe        | john.doe@example.com | 2023-01-15
(1 row)

postgres=# INSERT INTO employees (first_name, last_name, email, hire_date)
VALUES ('Patrick', 'r', 'patrick.r@example.com', '2023-09-01');
INSERT 0 1
postgres=# 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)

postgres=#
```

```
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

A screenshot of a terminal window titled "postgres-test" running on a Mac OS X system. The window includes standard OS X window controls (red, yellow, green buttons) and a dock icon for "postgres-test". The terminal interface has tabs for PROBLEMS, TERMINAL, OUTPUT, DEBUG CONSOLE, PORTS, and COMMENTS. A sidebar on the left contains icons for file operations like copy, paste, search, and navigation.

```
postgres=# SELECT * FROM employees;
 id | first_name | last_name | email | hire_date
----+-----+-----+-----+
(0 rows)

postgres=# INSERT INTO employees (first_name, last_name, email, hire_date)
VALUES ('John', 'Doe', 'john.doe@example.com', '2023-01-15');
INSERT 0 1
postgres=# SELECT * FROM employees;
 id | first_name | last_name | email | hire_date
----+-----+-----+-----+
 1 | John       | Doe        | john.doe@example.com | 2023-01-15
(1 row)

postgres=# INSERT INTO employees (first_name, last_name, email, hire_date)
VALUES ('Patrick', 'r', 'patrick.r@example.com', '2023-09-01');
INSERT 0 1
postgres=# 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)

postgres=# SELECT first_name, last_name FROM employees;
 first_name | last_name
-----+-----
 John      | Doe
 Patrick   | r
(2 rows)

postgres=#

```

The terminal shows the execution of several PostgreSQL commands. It starts by querying the "employees" table, which currently has 0 rows. Then, it inserts a new row for "John Doe" with an email of "john.doe@example.com" and a hire date of "2023-01-15". After the insert, it queries the table again, showing the new row. Next, it inserts another row for "Patrick r" with an email of "patrick.r@example.com" and a hire date of "2023-09-01". Finally, it queries the table again, showing both rows.

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

Server-Side Data Management

Postgres Database Operations

The screenshot shows a PostgreSQL terminal window titled "postgres-test". The interface includes a toolbar with icons for file operations, search, and Docker integration. The main area displays the following PostgreSQL session:

```
postgres=# INSERT INTO employees (first_name, last_name, email, hire_date)
VALUES ('Patrick', 'r', 'patrick.r@example.com', '2023-09-01');
INSERT 0 1
postgres=# 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)

postgres=# SELECT first_name, last_name FROM employees;
 first_name | last_name
-----+-----
 John       | Doe
 Patrick    | r
(2 rows)

postgres=# UPDATE employees
SET email = 'john.newemail@example.com'
WHERE id = 1;
UPDATE 1
postgres=# SELECT * FROM employees;
 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)
```

On the right side of the terminal, there is a purple text overlay containing the following SQL command:

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

At the bottom of the terminal, there is a magenta text overlay containing the following SQL command:

```
SELECT * FROM employees;
```

The bottom status bar shows the current session details: Ln 1, Col 21, Spaces: 4, UTF-8, LF, Docker, Go Live, and a small Docker icon.

SELECT * FROM employees;

Server-Side Data Management

Postgres Database Operations

The screenshot shows a PostgreSQL terminal window titled "postgres-test". The interface includes a toolbar with icons for file operations, search, and Docker integration. The main area displays the following SQL queries and their results:

```
PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE PORTS COMMENTS
1 | John      | Doe      | john.doe@example.com | 2023-01-15
2 | Patrick   | r        | patrick.r@example.com | 2023-09-01
(2 rows)

postgres=# SELECT first_name, last_name FROM employees;
 first_name | last_name
-----+-----
 John       | Doe
 Patrick    | r
(2 rows)

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

postgres=# SELECT * FROM employees;
 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)

postgres=# DELETE FROM employees
WHERE id = 1;
DELETE 1

postgres=# SELECT * FROM employees;
 id | first_name | last_name |          email           | hire_date
----+-----+-----+-----+-----+
 2 | Patrick   | r        | patrick.r@example.com | 2023-09-01
(1 row)

postgres=|
```

The bottom status bar indicates the cursor is at "Ln 1, Col 21" with "Spaces: 4" and encoding "UTF-8". It also shows "LF" (Line Feed), an empty brace {}, "Docker" integration, and "Go Live" status.

```
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 localhost
  port: 5432,
  user: 'postgres',           // default user
  password: '12345',          // password set in the container command
  database: 'postgres',        // default database
});
```

Server-Side Data Management

Node Database Connection

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 connection

```
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();

4. Run dbtest locally

```
node dbtest.js
```

The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows a project structure for "POSTGRES-TEST" containing "dbsetup", "node_modules", "dbtest.js", "Dockerfile", "package-lock.json", and "package.json".
- Code Editor:** The active file is "dbtest.js". The code uses `async/await` to connect to a PostgreSQL database.
- Terminal:** Shows the command "node dbtest.js" being run and the output "Connected to PostgreSQL database with async/await".
- Status Bar:** Displays file information like "Ln 12, Col 27", "Spaces: 4", "UTF-8", "LF", and "JavaScript".

```
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 localhost
  port: 5432,
  user: 'postgres',           // default user
  password: '12345',          // password set in the container command
  database: 'postgres',        // default database
});

// Connect to the database
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
```

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 function selectRecords() {
  const result = await dbClient.query('SELECT * FROM users');
  return result.rows;
}

const dbtest = {
  dbClient,
  selectRecords,
};

module.exports = dbtest;
```

Rerun dbtest.js

The screenshot shows a VS Code interface with the following details:

- Explorer View:** Shows a project structure under "POSTGRES-TEST" containing "dbtest.js", "Dockerfile", "package-lock.json", and "package.json".
- Code Editor:** The file "dbtest.js" is open, displaying the following code:

```
13  async function connectDB() {
14    try {
15      await client.connect();
16      console.log('Connected to PostgreSQL database with async/await');
17    } catch (err) {
18      console.error('Connection error', err.stack);
19    }
20  }
21 connectDB(); // should be called before any other function
22
23 async function selectRecords() {
24   const selectQuery = 'SELECT * FROM employees;';
25   try {
26     const res = await client.query(selectQuery);
27     console.log('All employees:', res.rows);
28   } catch (err) {
29     console.error('Error selecting records', err.stack);
30   }
31 }
32 selectRecords()
```

- Terminal:** Shows the output of the database query:

```
first_name: 'Martin',
last_name: 'K',
email: 'martin.k@example.com',
hire_date: 2023-01-14T23:00:00.000Z
},
{
  id: 2,
  first_name: 'Jerome',
  last_name: 'L',
  email: 'newmail.landre@example.com',
  hire_date: 2023-01-14T23:00:00.000Z
}
```

- Status Bar:** Shows file statistics: Ln 32, Col 1, Spaces: 4, UTF-8, LF, {}, JavaScript, Go Live.

Server-Side Data Management

Node Database Connection

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!

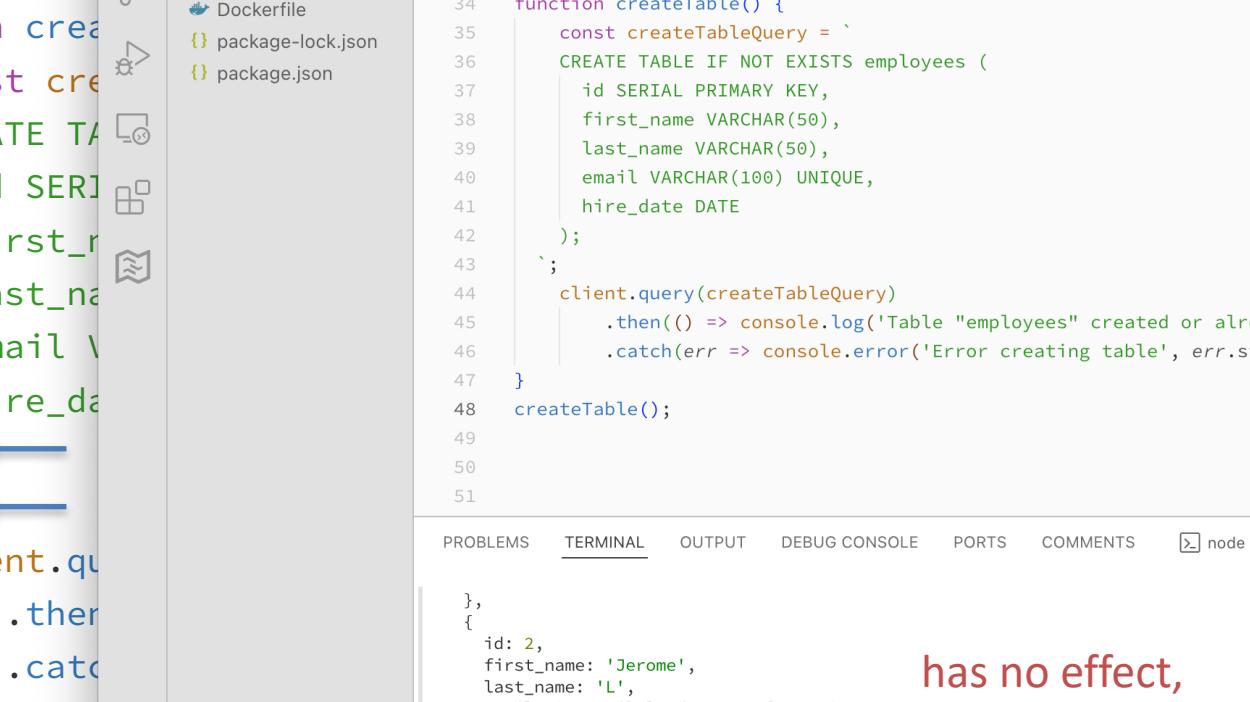
has no effect,
if the table is already there

Rerun dbtest.js

Server-Side Data Management

Node Database Connection

6. Create a table



```
// At first time
// Multiline
function createTable() {
    const client = new Client();
    client.connect();
    const query = `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
    );`;
    client.query(query)
        .then(() => console.log('Table "employees" created or already exists'))
        .catch(err => console.error('Error creating table', err.stack));
}
createTable();

// dbtest.js
// Dockerfile
// package-lock.json
// package.json
```

PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE PORTS COMMENTS node + × ⌂ ⌂ ...

```
},
{
  id: 2,
  first_name: 'Jerome',
  last_name: 'L',
  email: 'newmail.landre@example.com',
  hire_date: 2023-01-14T23:00:00.000Z
}]^C
```

has no effect,
if the table is already there

has no effect,
if the table is already there

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) ← Placeholder for actual values
    RETURNING *;
  `;
  client.query(insertQuery, insertValues) ← Actual values in array, sec. para
    .then(res => console.log('Inserted record:', res.rows[0]))
    .catch(err => console.error('Error inserting record', err.stack));
}

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

Rerun dbtest.js

Server-Side Data Management

Node Database Connection

7. Insert

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- Title Bar:** postgres-test
- File Explorer (Left):** Shows a project structure under "POSTGRES-TEST" containing "dbsetup", "node_modules", "dbtest.js", "Dockerfile", "package-lock.json", and "package.json".
- Code Editor (Center):** Displays the content of "dbtest.js".

```
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.yourlastname@example.com', '2023-01-15'];
insertRecord(insertValues);
const insertValues2 = ['name', 'lastname', 'name.lastname@example.com', '2023-01-15'];
insertRecord(insertValues2);
selectRecords()
```
- Terminal (Bottom):** Shows the output of the code execution:

```
first_name: 'You',
last_name: 'YourLastame',
email: 'you.yourlastname@example.com',
hire_date: 2023-01-14T23:00:00.000Z
},
{
  id: 13,
  first_name: 'name',
  last_name: 'lastname',
  email: 'name.lastname@example.com',
  hire_date: 2023-01-14T23:00:00.000Z
}
```
- Status Bar (Bottom):** Shows the current file is "dbtest.js", line 67, column 16, with 4 spaces, encoding is UTF-8, line feed is LF, and the status "JavaScript Go Live".

Server-Side Data Management

Node Database Connection

8. Select records (value) by value

```
async function getEmailID(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);
  }
}

getEmailID('patrick.r@example.com').then((id) => {
  console.log('ID:', id);                                ID: 7
})
```

Rerun dbtest.js

Server-Side Data Management

Node Database Connection

8. Select

asyn

get]

)

Rerun

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows a project structure under "POSTGRES-TEST" containing "dbsetup", "node_modules", "dbtest.js" (selected), "Dockerfile", "package-lock.json", and "package.json".
- Code Editor:** The "dbtest.js" file is open, displaying the following code:

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

// const insertValues2 = ['name', 'lastname', 'name.lastname@example.com', '2023-01-15'];
// insertRecord(insertValues2);

// selectRecords()

async function getEmailID(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);
  }
}

getEmailID('patrick.r@example.com').then((id) => {
  console.log('ID:', id);
})
```
- Terminal:** The terminal tab is active, showing the command `node dbtest.js` and its output:

```
Connected to PostgreSQL database with async/await
ID: 7
```
- Status Bar:** Shows "Ln 81, Col 1" and "Spaces: 4" along with other standard status bar icons.

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));
}

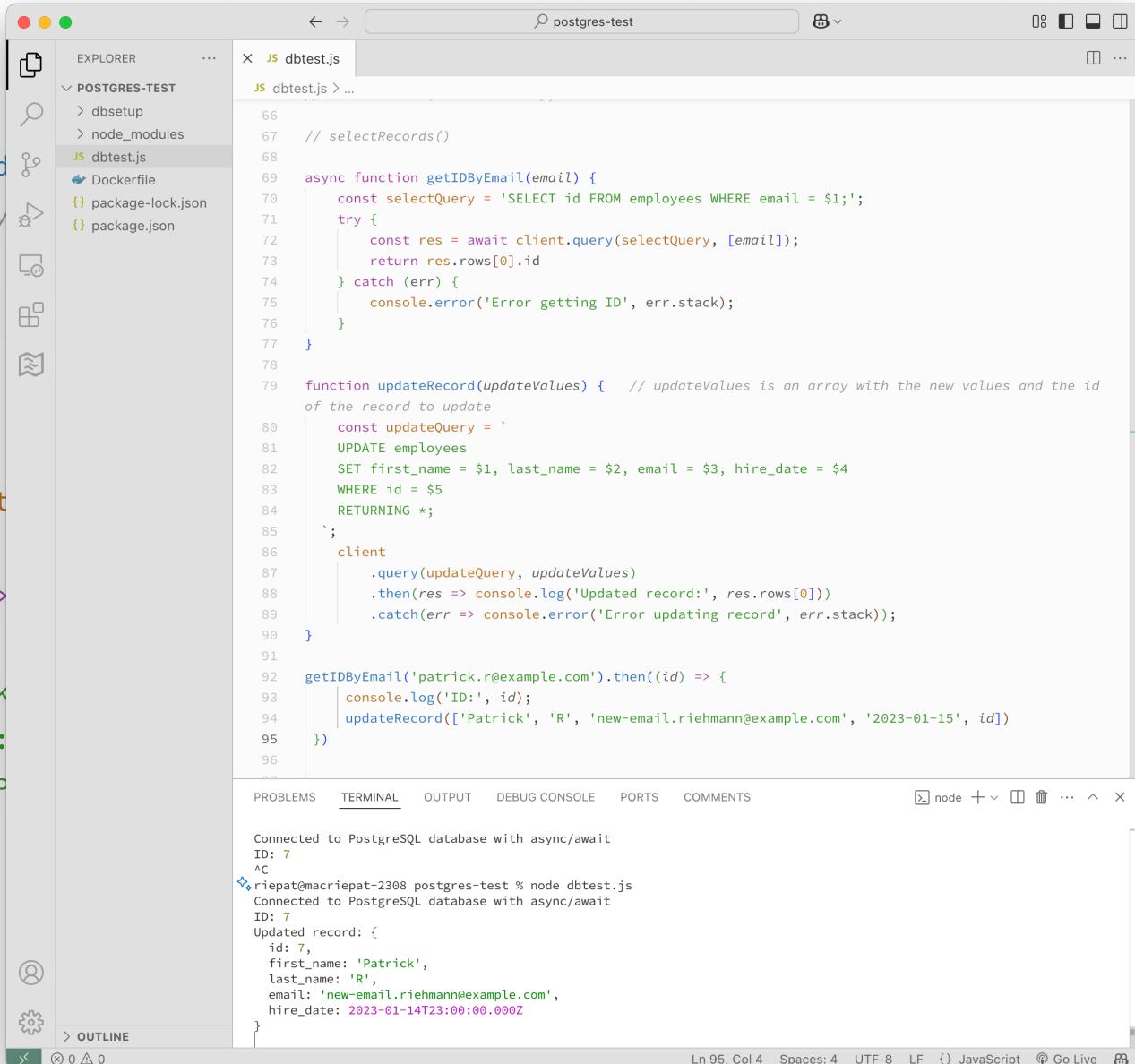
getIDByEmail('patrick.r@example.com').then((id) => {
  console.log('ID:', id);
  updateRecord(['Patrick', 'R', 'new-email.riehmann@example.com', '2023-01-15', id])
})
```

Server-Side Data Management

Node Database Connection

9. Update records

```
function updateRecord() {
  const updateQuery = `
    UPDATE employees
    SET first_name =
      WHERE id = $1
    RETURNING *;
  `;
  client.query(updateQuery)
    .then(res =>
      .catch(err =>
    })
  getIDByEmail('patrick.riehmann@example.com')
    console.log('ID:', res.rows[0].id)
    updateRecord(['Patrick', 'R', 'new-email.riehmann@example.com', '2023-01-15', res.rows[0].id])
  })
}
```



```
Connected to PostgreSQL database with async/await
ID: 7
^C
riepat@macriepat-2308 postgres-test % node dbtest.js
Connected to PostgreSQL database with async/await
ID: 7
Updated record:
  id: 7,
  first_name: 'Patrick',
  last_name: 'R',
  email: 'new-email.riehmann@example.com',
  hire_date: 2023-01-14T23:00:00.000Z
```

Server-Side Data Management

Node Database Connection

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

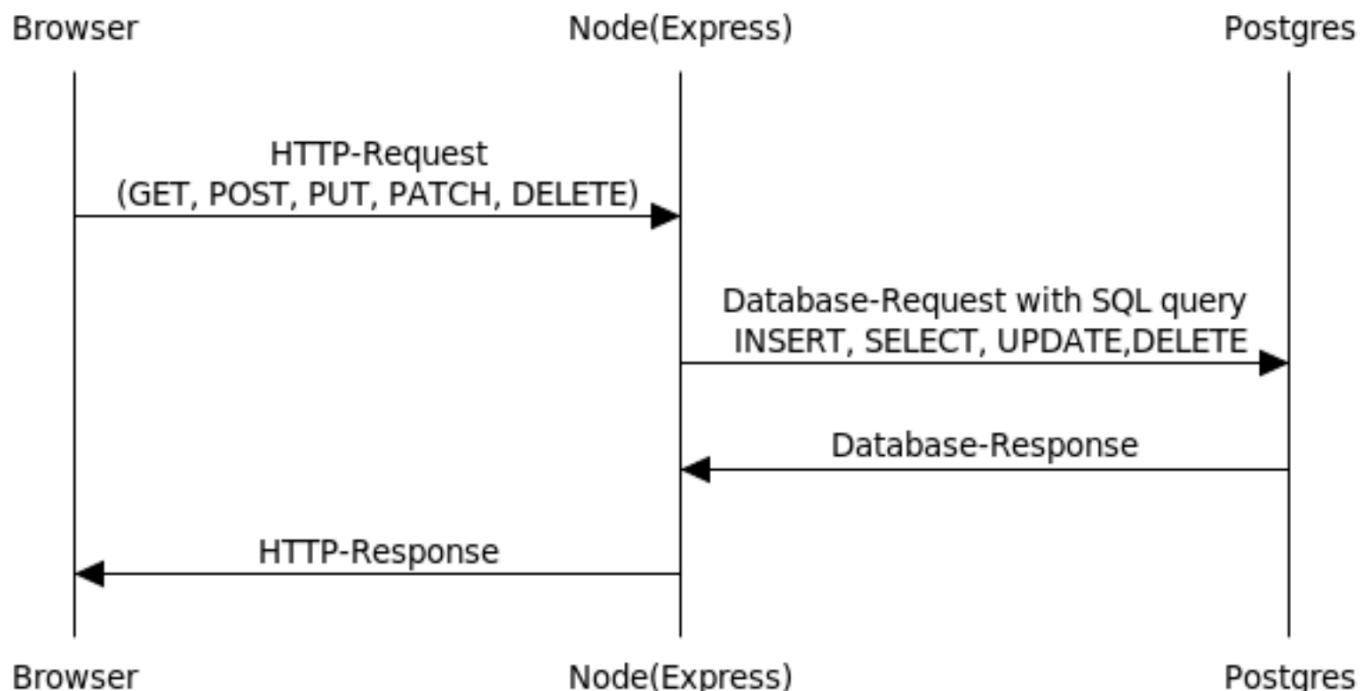
Express Database Minimal Passthrough Example - Hint Project

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);
  }
});
```

Route definition

Database transaction

Response sending

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

Database config

```
// 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();
```

Database connection

Server listen on port

Server-Side Data Management

Express Database Minimal Passthrough Example

A screenshot of a web browser window titled "postgres-test" showing the URL "localhost:3000/all". The browser interface includes standard controls like back, forward, and search, along with developer tools icons.

The main content area displays a JSON array of five objects, each representing a database record with fields: id, first_name, last_name, email, and hire_date.

```
[{"id": 3, "first_name": "Jerome", "last_name": "Landre", "email": "jerome.landre@example.com", "hire_date": "2023-01-14T23:00:00.000Z"}, {"id": 6, "first_name": "Jerome", "last_name": "L", "email": "jerome.l@example.com", "hire_date": "2023-01-14T23:00:00.000Z"}, {"id": 7, "first_name": "Patrick", "last_name": "R", "email": "patrick.r@example.com", "hire_date": "2023-01-14T23:00:00.000Z"}, {"id": 8, "first_name": "Martin", "last_name": "K", "email": "martin.k@example.com", "hire_date": "2023-01-14T23:00:00.000Z"}, {"id": 2, "first_name": "Jerome"}]
```

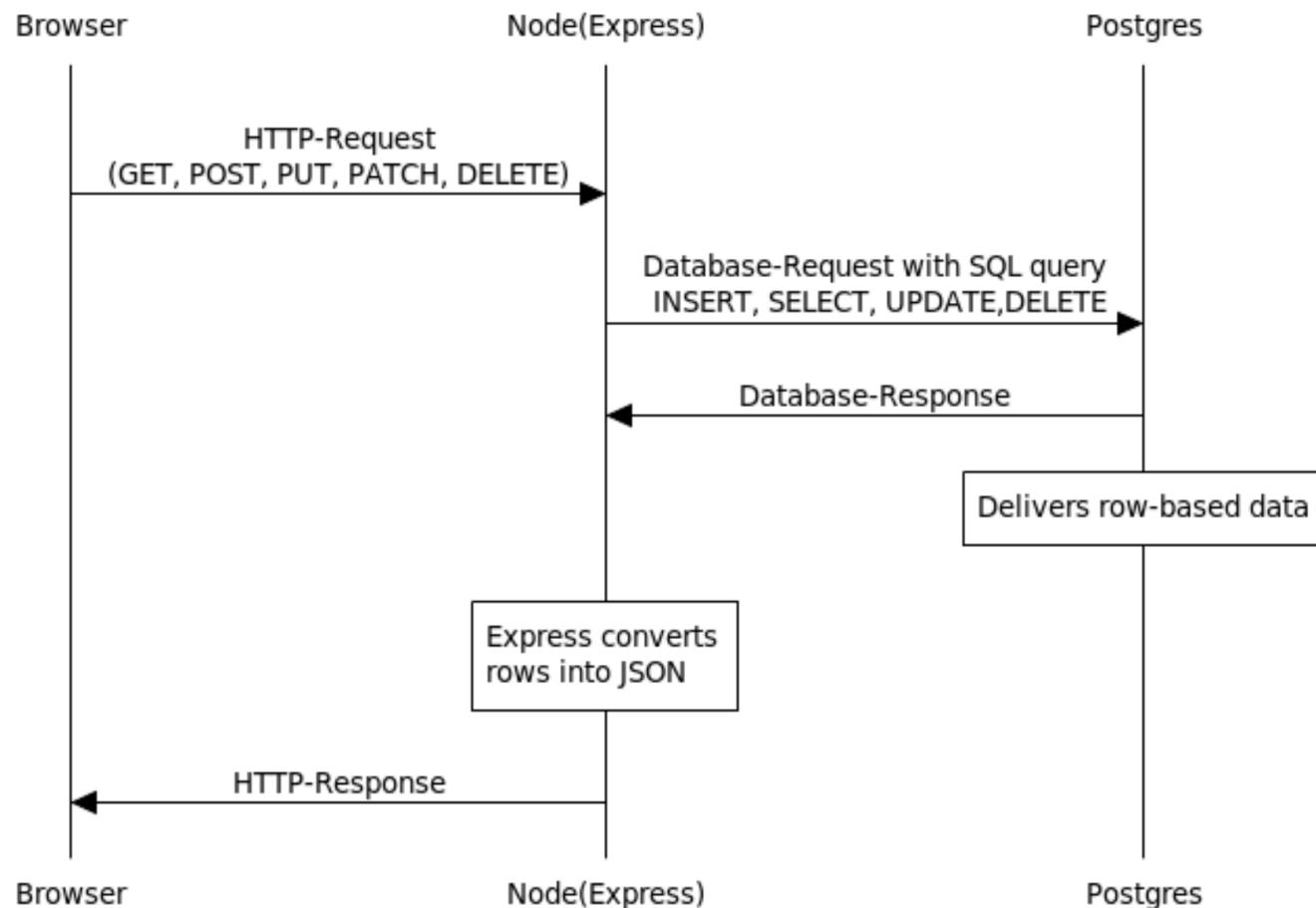
At the bottom of the browser window, there is a developer tool sidebar with tabs for "OUTLINE" and "CONSOLE". The "CONSOLE" tab shows the same JSON array again, indicating it is being evaluated or displayed in the developer tools.

Page footer: Ln 20, Col 34 Spaces: 4 UTF-8 LF {} JavaScript ⚡ Go Live

Server-Side Data Management

Express Database Minimal Passthrough Example - Hint Project

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



Server-Side Data Management

Express Database Minimal Passthrough Example - Hint Project

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

