An **asynchronous function** in JavaScript is a function that operates asynchronously, meaning it can perform tasks in the background without blocking the execution of other code. This is particularly useful for tasks that take time to complete, such as making network requests, reading files, or querying a database.

### Key Features of Asynchronous Functions:

**Non-blocking**: Asynchronous functions allow other code to run while waiting for a long-running operation to complete.

async**Keyword**: A function is marked as asynchronous by using the async keyword before the function declaration.

await**Keyword**: Inside an async function, you can use the await keyword to pause the execution of the function until a promise is resolved (i.e., the asynchronous operation completes).

async function fetchData() {

// Simulate an asynchronous operation (e.g., fetching data from a server)

const response = await fetch("https://api.example.com/data");

const data = await response.json();

return data;

}

The fetchData function is marked as async.

The await keyword is used to wait for the fetch call to complete and then for the response.json() call to complete.

The function returns the resolved value of the promise.

pp.get("/", async (req, res) => {

const result = await db.query("SELECT country\_code FROM visited\_countries");

// ...

});

async: The function is marked as async, allowing the use of await inside it.

await db.query(...): The await keyword pauses the function's execution until the database query completes and returns a result. During this time, other code outside this function can continue running.

Non-blocking: While waiting for the database query, the server can handle other requests or tasks.

Why Use Asynchronous Functions?

Efficiency: Asynchronous functions prevent blocking the main thread, making applications more responsive.

Simpler Code: Using async/await makes asynchronous code look and behave more like synchronous code, making it easier to read and maintain.

Error Handling: You can use try-catch blocks to handle errors in asynchronous code, just like in synchronous code.

pp.get("/", async (req, res) => {

try {

const result = await db.query("SELECT country\_code FROM visited\_countries");

let countries = result.rows.map((row) => row.country\_code);

res.render("index.ejs", { countries, total: countries.length });

} catch (error) {

console.error("Error fetching data:", error);

res.status(500).send("Internal Server Error");

} finally {

db.end();

}

});

An asynchronous function is a powerful tool in JavaScript for handling tasks that take time to complete, such as database queries, API calls, or file operations. By using async/await, you can write clean, non-blocking code that is easier to understand and maintain.

In **PostgreSQL**, tables are created using the CREATE TABLE statement, and each column in a table has a specific **data type**. PostgreSQL provides a wide range of data types and **SQL keywords** to define, manipulate, and query tables.

CREATE TABLE users (

id SERIAL PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL,

age INT CHECK (age >= 18),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

· SERIAL → Auto-increment integer.

· VARCHAR(n) → Variable-length string with a maximum of n characters.

· UNIQUE → Ensures unique values in the column.

· CHECK (age >= 18) → Adds a constraint to allow only values >= 18.

· TIMESTAMP DEFAULT CURRENT\_TIMESTAMP → Stores the date/time when the row is inserted.

## ****2. Data Types in PostgreSQL****

### ****Numeric Data Types****

| **Data Type** | **Description** |
| --- | --- |
| SMALLINT | 2-byte integer (-32,768 to 32,767) |
| INTEGER (INT) | 4-byte integer (-2 billion to 2 billion) |
| BIGINT | 8-byte integer |
| DECIMAL(p, s) | Fixed-point number with precision p and scale s |
| NUMERIC(p, s) | Same as DECIMAL, preferred for exact calculations |
| REAL | 4-byte floating-point number |
| DOUBLE PRECISION | 8-byte floating-point number |
| SERIAL | Auto-incrementing 4-byte integer |
| BIGSERIAL | Auto-incrementing 8-byte integer |

### ****String Data Types****

| **Data Type** | **Description** |
| --- | --- |
| CHAR(n) | Fixed-length string of n characters |
| VARCHAR(n) | Variable-length string up to n characters |
| TEXT | Unlimited-length string |

### ****Date/Time Data Types****

| **Data Type** | **Description** |
| --- | --- |
| DATE | Stores date (YYYY-MM-DD) |
| TIME | Stores time (HH:MM:SS) |
| TIMESTAMP | Date + time without time zone |
| TIMESTAMPTZ | Date + time with time zone |
| INTERVAL | Stores a time span (e.g., INTERVAL '2 hours') |

### ****Boolean Data Type****

| **Data Type** | **Description** |
| --- | --- |
| BOOLEAN | Can store TRUE, FALSE, or NULL |

### ****JSON & Array Types****

| **Data Type** | **Description** |
| --- | --- |
| JSON | Stores JSON data |
| JSONB | Optimized binary JSON (faster than JSON) |
| ARRAY | Stores an array of elements (e.g., INTEGER[]) |

CREATE TABLE orders (

id SERIAL PRIMARY KEY,

products TEXT[],

details JSONB

);

## ****3. Important PostgreSQL Keywords****

### ****DDL (Data Definition Language) - Creating & Modifying Tables****

| **Keyword** | **Description** |
| --- | --- |
| CREATE TABLE | Creates a new table |
| DROP TABLE | Deletes a table |
| ALTER TABLE | Modifies an existing table |
| TRUNCATE | Deletes all records from a table |

### ****ML (Data Manipulation Language) - Inserting & Updating Data****

| **Keyword** | **Description** |
| --- | --- |
| INSERT INTO | Adds a new record |
| UPDATE | Modifies existing records |
| DELETE FROM | Removes records |
| SELECT | Retrieves data |

INSERT INTO users (name, email, age) VALUES ('John Doe', 'john@example.com', 25);

UPDATE users SET age = 26 WHERE id = 1;

DELETE FROM users WHERE id = 1;

### ****DQL (Data Query Language) - Retrieving Data****

| **Keyword** | **Description** |
| --- | --- |
| SELECT | Retrieves data |
| WHERE | Filters records |
| ORDER BY | Sorts results |
| GROUP BY | Groups results |
| HAVING | Filters grouped records |
| JOIN | Combines data from multiple tables |
|  |  |

SELECT name, email FROM users WHERE age > 18 ORDER BY name;

### ****oins in PostgreSQL****

| **Type of Join** | **Description** |
| --- | --- |
| INNER JOIN | Returns only matching records |
| LEFT JOIN | Returns all records from the left table and matching ones from the right |
| RIGHT JOIN | Returns all records from the right table and matching ones from the left |
| FULL JOIN | Returns all records from both tables |

SELECT users.name, orders.id

FROM users

INNER JOIN orders ON users.id = orders.user\_id;

## ****4. Constraints in PostgreSQL****

| **Constraint** | **Description** |
| --- | --- |
| PRIMARY KEY | Uniquely identifies each row |
| FOREIGN KEY | Links tables together |
| UNIQUE | Ensures column values are unique |
| CHECK | Validates data conditions |
| NOT NULL | Prevents NULL values |

CREATE TABLE orders (

id SERIAL PRIMARY KEY,

user\_id INT REFERENCES users(id),

total\_price DECIMAL CHECK (total\_price > 0)

);

## ****5. Indexes in PostgreSQL****

Indexes improve query performance.

CREATE INDEX idx\_email ON users(email);

## ****6. Views in PostgreSQL****

A view is a virtual table based on a query.

CREATE VIEW active\_users AS

SELECT id, name, email FROM users WHERE active = TRUE;

### ****7. Stored Procedures & Functions****

Stored procedures contain multiple SQL statements

CREATE FUNCTION get\_user\_count() RETURNS INT AS $$

BEGIN

RETURN (SELECT COUNT(\*) FROM users);

END;

$$ LANGUAGE plpgsql;

The SERIAL keyword in PostgreSQL is a special data type that automatically generates unique integer values for a column. It is commonly used for **auto-incrementing primary keys**.

CREATE TABLE capitals (

id SERIAL PRIMARY KEY, -- Auto-incrementing unique identifier

name VARCHAR(45), -- Name of the capital (max 45 characters)

flag TEXT -- Stores flag data (could be a URL or Base64 string)

);

CHECK (flag ~\* '^https?://')

READ DATA :

Select \* from <mytable>

To download you just write npm I pg