

Taller NodeJS con Express y PostgreSQL en SupaBase

JORGE ANDRES CASTRO PACHON

ID:827833

BASES DE DATOS MASIVAS

DOCENTE:

WILLIAM ALEXANDER MATALLANA

CORPORACIÓN UNIVERSITARIA MINUTO DE

DIOS

INGENERIA DE SISTEMAS

ZIPQUIRÁ, CUNDINAMARCA 2025

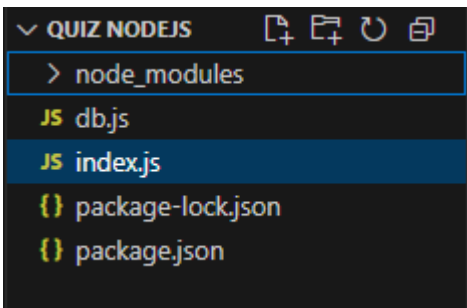
Express

1. Inicializar proyecto

Aquí instalamos las dependencias necesarias, que van a ser: Express (el framework) Pg (Postgres para supabase), dotenv (para variables de entorno), cors (para exponer las api si las usamos en otro framework de front-end)

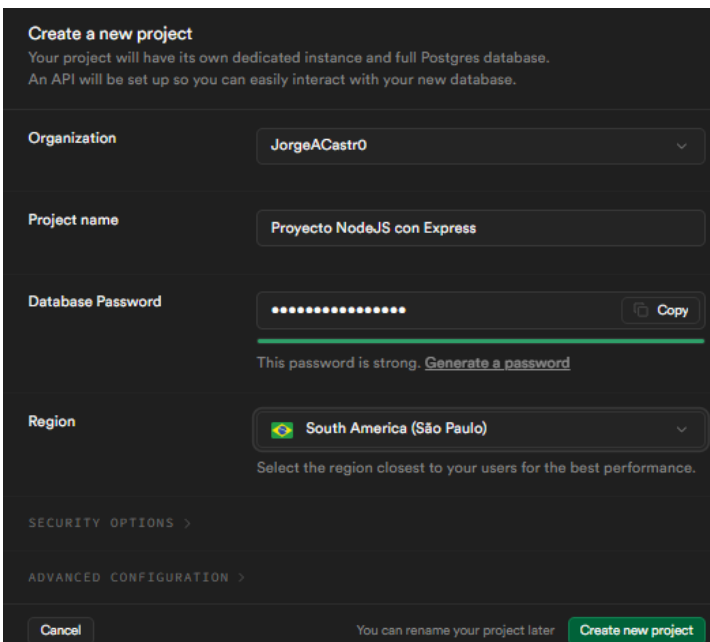
```
PS C:\Users\SDA-48-239\OneDrive - uniminuto.edu\6to Semestre\Bases de datos Masivas\Quiz NodeJS> npm install express pg dotenv cors
added 83 packages in 22s

15 packages are looking for funding
  run `npm fund` for details
PS C:\Users\SDA-48-239\OneDrive - uniminuto.edu\6to Semestre\Bases de datos Masivas\Quiz NodeJS> |
```

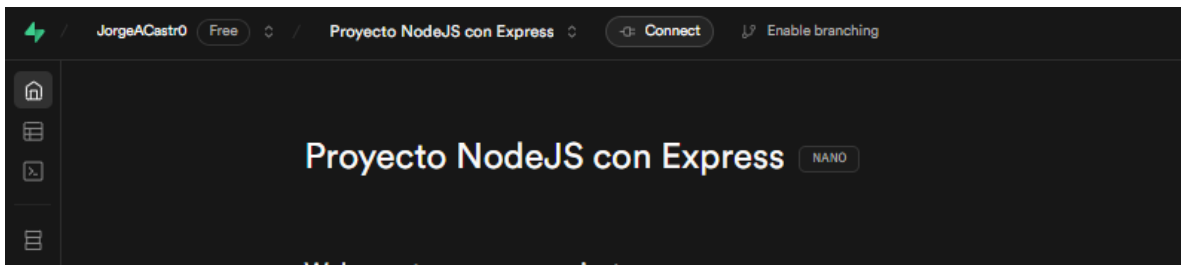


2. Creación de la DB en Supabase

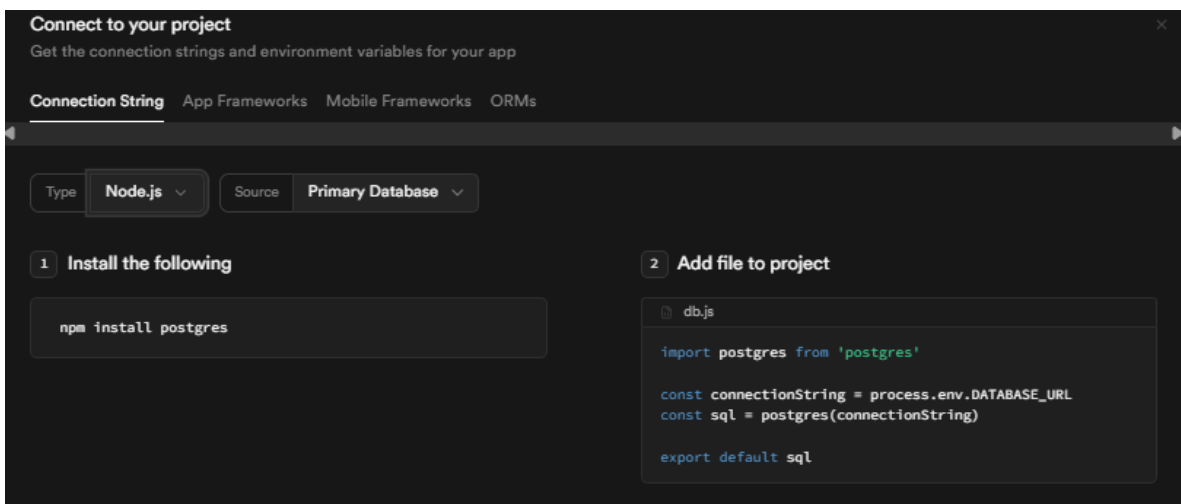
Ahora vamos a crear el proyecto nuevo en Supabase

A screenshot of the Supabase 'Create a new project' form. The form has a dark theme. At the top, it says 'Create a new project' and provides a brief description. Below this are several input fields: 'Organization' with the value 'JorgeACastro0', 'Project name' with the value 'Proyecto NodeJS con Express', 'Database Password' which is masked with dots and has a 'Copy' button, and 'Region' with the value 'South America (São Paulo)'. There are also links for 'SECURITY OPTIONS' and 'ADVANCED CONFIGURATION'. At the bottom, there are 'Cancel' and 'Create new project' buttons. A note at the bottom says 'You can rename your project later.'

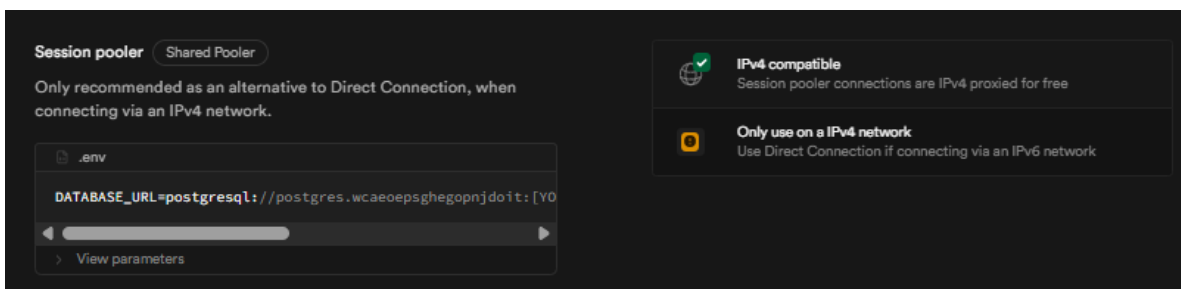
Ahora vamos a “Connect” para averiguar



Aquí seguiremos los pasos para hacer la conexión a Supa



Usaremos el Session Pooler para conectar el proyecto, esta url la pondremos en nuestro archivo db.js



En nuestro db.js pondremos esta línea

```
JS db.js > ...
1 import postgres from 'postgres';
2
3 const sql = postgres('postgres://postgres.wcaeoepsghogopnjdoit:[YOUR-PASSWORD]@aws-0-sa-east-1.pooler.supabase.co
4 |
5 export default sql;
```

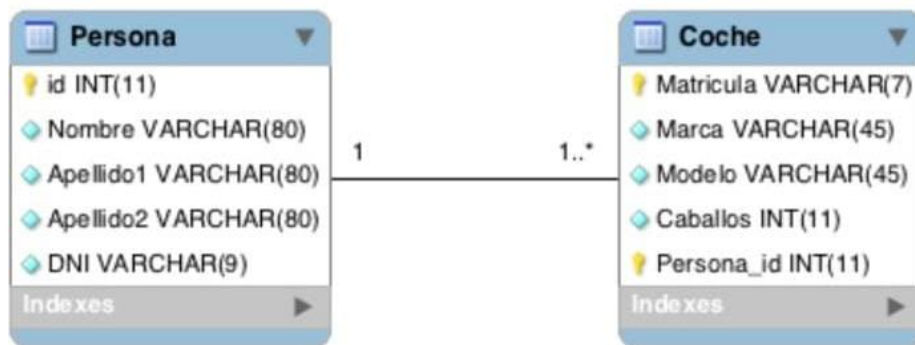
Pero vamos a reemplazar donde dice [YOUR PASSWORD] por la contraseña que pusimos al principio del proyecto, quedaría así:

```
import postgres from 'postgres';

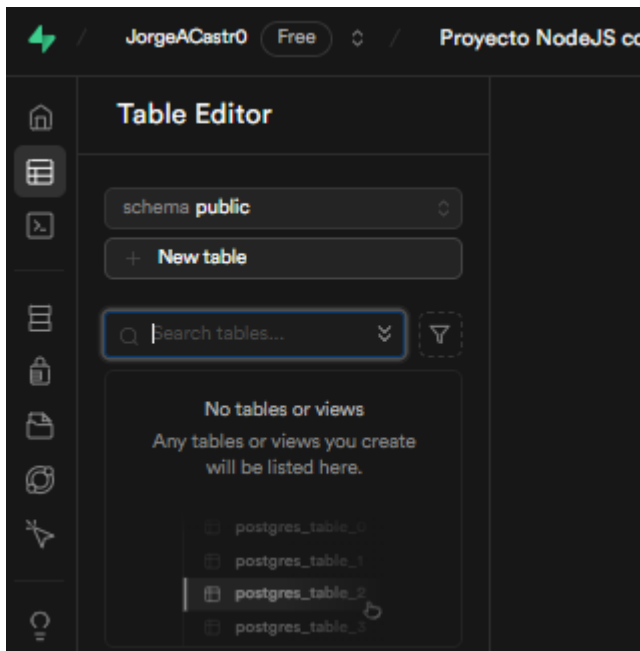
const sql = postgres('postgres://postgres.wcaeoepsghogopnjdoit:oUazj1LV6Q6ZT95p@aws-0-sa-east-1.pooler.supabase.com:5432/postgres');

export default sql;
```

Ya teniendo la conexión a la base de datos, vamos a crear las tablas



Vamos a supabase y vamos a la sección de table editor



Creamos una nueva tabla:

The screenshot shows a 'Create a new table under public' form. The form has two main input fields: 'Name' with the value 'Persona' and 'Description' with the value 'Tabla persona'. Below these fields, there is a checkbox labeled 'Enable Row Level Security (RLS)' which is checked with a green checkmark. To the right of the checkbox is a 'Recommended' badge. Below the checkbox, there is a line of text: 'Restrict access to your table by enabling RLS and writing Postgres policies'.

Asignamos las columnas:

Name	Type	Default Value	Primary
id	# int8	NULL	<input checked="" type="checkbox"/>
Nombre	T varchar	NULL	<input type="checkbox"/>
Apellido1	T varchar	NULL	<input type="checkbox"/>
Apellido2	T varchar	NULL	<input type="checkbox"/>
DNI	T varchar	NULL	<input type="checkbox"/>

[Add column](#)

Después de tener la primera tabla, crearemos la segunda en donde pondremos la llave foránea que se necesita

Create a new table under public

Name: Coche

Description: Tabla coche con llave foranea en el id Persona

☒ Enable Row Level Security (RLS) Recommended
Restrict access to your table by enabling RLS and writing Postgres policies.

Creamos las columnas de la segunda tabla:

Name	Type	Default Value	Primary
Matricula	# int8	NULL	<input checked="" type="checkbox"/>
Marca	T varchar	NULL	<input type="checkbox"/>
Modelo	T varchar	NULL	<input type="checkbox"/>
Caballos	# int8	NULL	<input type="checkbox"/>
Persona_id	int8	NULL	<input type="checkbox"/>

Add column

Ahora crearemos la llave foránea necesaria

public

Select a table to reference to

Persona

Select columns from public.Personato reference to

public.Coche public.Persona

Persona_id → id

Add another column

Column types will be updated

The following columns will have their types updated to match their referenced column

- Persona_id → int8

Which action is most appropriate?

Action if referenced row is updated

Cascade

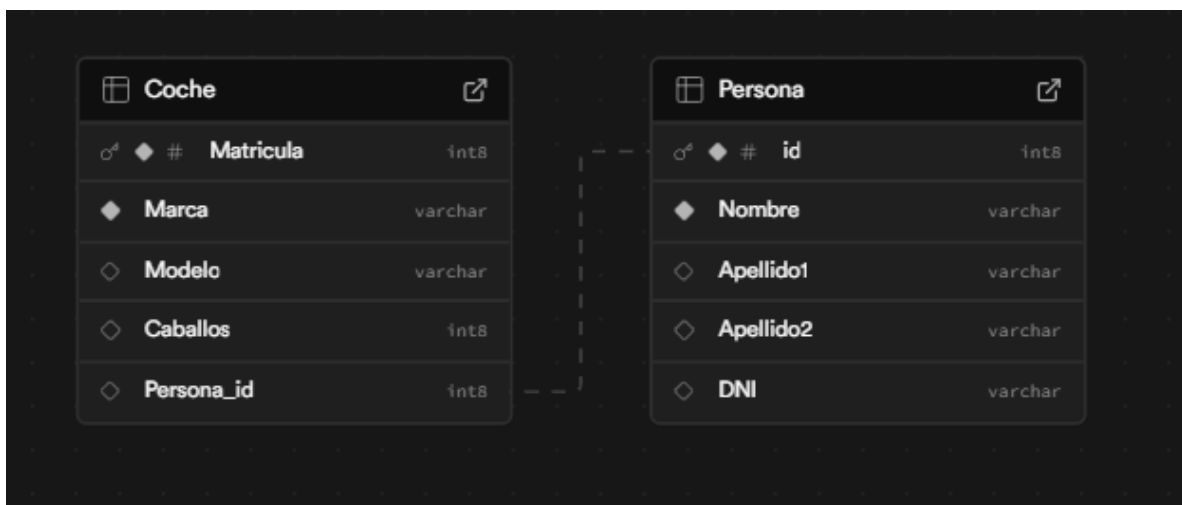
Cascade: Updating a record from public.Persona will also update any records that reference it in this table

Action if referenced row is removed

Cascade

Cascade: Deleting a record from public.Persona will also delete any records that reference it in this table

Después de tener las tablas creadas vamos a ver el modelo relacional que nos da Supabase y compararemos:



Muy bien ahora nos encargaremos de insertar 100 registros en nuestras tablas, lo haremos con el editor SQL de SupaBase y con ayuda de la IA generativa para mayor velocidad.

Le pasaremos este prompt a chatGPT: “hola, con base a la imagen del modelo relacional de mi base de datos ingresa 100 registros en cada tabla, puede ser que una persona tenga mas de un coche, pero un coche solo puede ser de una persona”

Nos arroja el siguiente resultado:

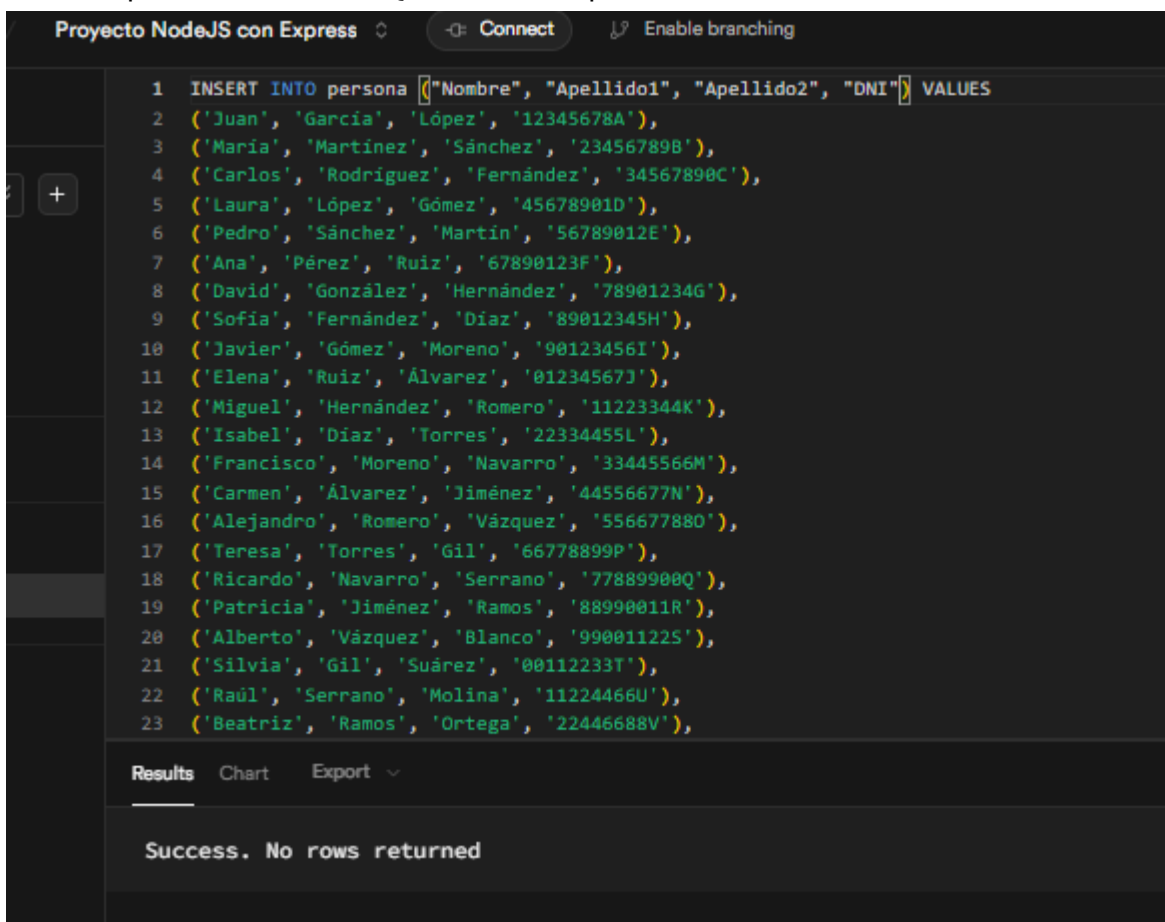
Para tabla Persona:

```
sql Copy
INSERT INTO Persona (Nombre, Apellido1, Apellido2, DNI) VALUES
('Juan', 'García', 'López', '12345678A'),
('María', 'Martínez', 'Sánchez', '23456789B'),
('Carlos', 'Rodríguez', 'Fernández', '34567890C'),
('Laura', 'López', 'Gómez', '45678901D'),
('Pedro', 'Sánchez', 'Martín', '56789012E'),
('Ana', 'Pérez', 'Ruiz', '67890123F'),
('David', 'González', 'Hernández', '78901234G'),
('Sofía', 'Fernández', 'Díaz', '89012345H'),
('Javier', 'Gómez', 'Moreno', '90123456I'),
('Elena', 'Ruiz', 'Álvarez', '01234567J'),
('Miguel', 'Hernández', 'Romero', '11223344K'),
('Isabel', 'Díaz', 'Torres', '22334455L'),
('Francisco', 'Moreno', 'Navarro', '33445566M'),
('Carmen', 'Álvarez', 'Jiménez', '44556677N'),
('Alejandro', 'Romero', 'Vázquez', '55667788O'),
('Teresa', 'Torres', 'Gil', '66778899P'),
('Ricardo', 'Navarro', 'Serrano', '77889900Q'),
```

Para tabla Coches:

```
INSERT INTO Coche (Matricula, Marca, Caballos, Persona_id) VALUES
('1234ABC', 'Toyota', 120, 1),
('2345BCD', 'Renault', 90, 1),
('3456CDE', 'Ford', 110, 2),
('4567DEF', 'Seat', 95, 3),
('5678EFG', 'Volkswagen', 105, 4),
('6789FGH', 'Peugeot', 115, 5),
('7890GHI', 'Citroën', 100, 6),
('8901HIJ', 'BMW', 180, 7),
('9012IJK', 'Mercedes', 200, 8),
('0123JKL', 'Audi', 170, 9),
('1122KLM', 'Hyundai', 130, 10),
('2233LMN', 'Kia', 125, 11),
('3344MNO', 'Nissan', 140, 12),
('4455NOP', 'Mazda', 150, 13),
('5566OPQ', 'Honda', 135, 14),
('6677PQR', 'Fiat', 85, 15),
```

Ahora lo pondremos en el SQLEditor de Supa Base:



```
1 INSERT INTO persona ("Nombre", "Apellido1", "Apellido2", "DNI") VALUES
2 ('Juan', 'Garcia', 'López', '12345678A'),
3 ('Maria', 'Martínez', 'Sánchez', '23456789B'),
4 ('Carlos', 'Rodríguez', 'Fernández', '34567890C'),
5 ('Laura', 'López', 'Gómez', '45678901D'),
6 ('Pedro', 'Sánchez', 'Martín', '56789012E'),
7 ('Ana', 'Pérez', 'Ruiz', '67890123F'),
8 ('David', 'González', 'Hernández', '78901234G'),
9 ('Sofía', 'Fernández', 'Díaz', '89012345H'),
10 ('Javier', 'Gómez', 'Moreno', '90123456I'),
11 ('Elena', 'Ruiz', 'Álvarez', '01234567J'),
12 ('Miguel', 'Hernández', 'Romero', '11223344K'),
13 ('Isabel', 'Díaz', 'Torres', '22334455L'),
14 ('Francisco', 'Moreno', 'Navarro', '33445566M'),
15 ('Carmen', 'Álvarez', 'Jiménez', '44556677N'),
16 ('Alejandro', 'Romero', 'Vázquez', '55667788O'),
17 ('Teresa', 'Torres', 'Gil', '66778899P'),
18 ('Ricardo', 'Navarro', 'Serrano', '77889900Q'),
19 ('Patricia', 'Jiménez', 'Ramos', '88990011R'),
20 ('Alberto', 'Vázquez', 'Blanco', '99001122S'),
21 ('Silvia', 'Gil', 'Suárez', '00112233T'),
22 ('Raúl', 'Serrano', 'Molina', '11224466U'),
23 ('Beatriz', 'Ramos', 'Ortega', '22446688V'),
```

Results Chart Export

Success. No rows returned

Al igual la tabla coche

```
1 INSERT INTO Coche (Matricula, Marca, Caballos, Persona_id) VALUES
2 ('1234ABC', 'Toyota', 120, 1),
3 ('2345BCD', 'Renault', 90, 1),
4 ('3456CDE', 'Ford', 110, 2),
5 ('4567DEF', 'Seat', 95, 3),
6 ('5678EFG', 'Volkswagen', 105, 4),
7 ('6789FGH', 'Peugeot', 115, 5),
8 ('7890GHI', 'Citroën', 100, 6),
9 ('8901HIJ', 'BMW', 180, 7),
10 ('9012IJK', 'Mercedes', 200, 8),
11 ('0123JKL', 'Audi', 170, 9),
12 ('1122KLM', 'Hyundai', 130, 10),
13 ('2233LMN', 'Kia', 125, 11),
14 ('3344MNO', 'Nissan', 140, 12),
15 ('4455NOP', 'Mazda', 150, 13),
16 ('5566OPQ', 'Honda', 135, 14),
17 ('6677PQR', 'Fiat', 85, 15),
18 ('7788QRS', 'Opel', 110, 16),
19 ('8899RST', 'Skoda', 120, 17),
20 ('9900STU', 'Volvo', 160, 18),
21 ('0011TUV', 'Mini', 120, 19),
22 ('1122UVW', 'Jeep', 190, 20),
23 ('2233VWX', 'Land Rover', 210, 21),
```

Results Chart Export ▾

Success. No rows returned

Ya teniendo construida la base de datos vamos a construir las apis para realizar un CRUD desde Node.js

Lo primero que vamos a hacer es crear un función que me permita probar si mi conexión a la base de datos esta bien

```
async function testConnection() {
  try {
    const result = await sql`SELECT 1`;
    console.log('✅ Conexión exitosa:', result);
  } catch (error) {
    console.error('❌ Error al conectar a la base de datos:', error);
  } finally {
    await sql.end(); // Cierra la conexión
  }
}

testConnection();
```

Que nos responde:

```
PS C:\Users\SDA-48-239\OneDrive - uniminuto.edu\6to Semestre\Bases de datos Masivas\Quiz NodeJS> node index.js
(node:3840) [MODULE_TYPELESS_PACKAGE_JSON] Warning: Module type of file:///C:/Users/SDA-48-239/OneDrive%20-%20uniminuto.edu/6to Semestre/Bases de datos Masivas/Quiz NodeJS/index.js is not specified and it doesn't parse as CommonJS.
Reparsing as ES module because module syntax was detected. This incurs a performance overhead.
To eliminate this warning, add "type": "module" to C:\Users\SDA-48-239\OneDrive - uniminuto.edu\6to Semestre\Bases de datos Masivas\Quiz NodeJS\package.json
(Use 'node --trace-warnings ...' to show where the warning was created)
El servidor esta corriendo
Conexión exitosa: Result(1) [ { '?column?': 1 } ]
```

Y en el index.js importo express y la conexión a la DB, al igual

```
JS index.js > ...
1  import express from 'express';
2  import sql from './db.js';
3
4  const app = express();
5
6  app.use(express.json());
7
8  app.use(express.urlencoded({extended:true}));
9
10 app.get('/api/prueba' , (req, res) => {
11   |
12   |   res.send('LA API FUNCIONA');
13   | });
14
15 //Crear puerto de conexion del servidor
16 const PORT = 3000;
17
18 //La conexion la va a escuchar por el puerto 3000 y si
19 app.listen(PORT, ()=>{
20   |   console.log('El servidor esta corriendo');
21   |
22   | });
```

Creo una api para consultar los datos de las tablas:

```
//API PARA VER LOS REGISTROS DE LAS TABLAS
app.get('/api/leerdatos', async (req, res) => {
  try {
    const result = await sql`SELECT * FROM persona `;

    res.status(200).json({
      success: true,
      message: "DATOS DE LA TABLA",
      data: result
    });

  } catch (error) {
    res.status(500).json({
      success: false,
      message: 'ERROR CONSULTANDO LA DB',
      details: error.message || 'ERROR DESCONOCIDO'
    });
  }
});
```

En postman la consumimos y nos muestra:

The screenshot shows the Postman interface for a GET request to `http://localhost:3000/api/leerdatos`. The response is a 200 OK status with a response time of 1.17 s and a size of 10.21 KB. The response body is displayed in JSON format:

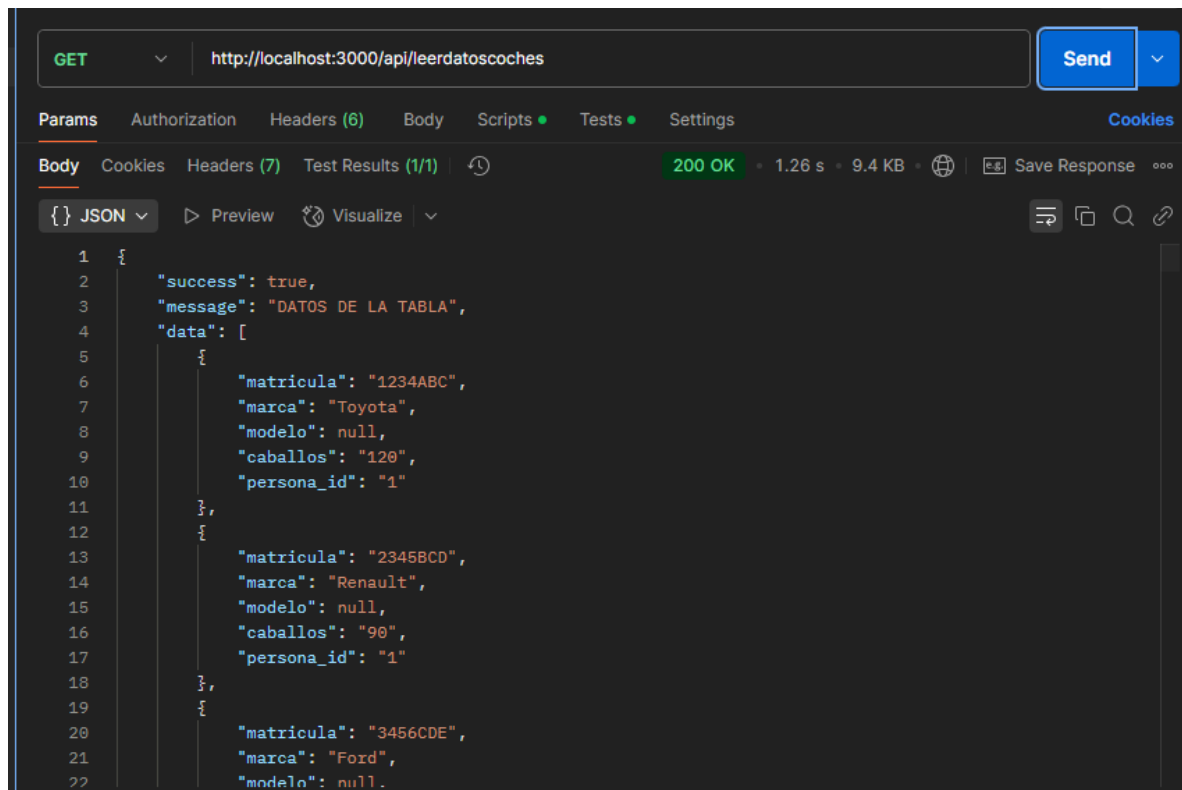
```
{
  "success": true,
  "message": "DATOS DE LA TABLA",
  "data": [
    {
      "id": "1",
      "Nombre": "Juan",
      "Apellido1": "García",
      "Apellido2": "López",
      "DNI": "12345678A"
    }
  ]
}
```

```

25     },
26     {
27         "id": "4",
28         "Nombre": "Laura",
29         "Apellido1": "López",
30         "Apellido2": "Gómez",
31         "DNI": "45678901D"
32     },
33     {
34         "id": "5",
35         "Nombre": "Pedro",
36         "Apellido1": "Sánchez",
37         "Apellido2": "Martín",
38         "DNI": "56789012E"
39     },
40     {
41         "id": "6",
42         "Nombre": "Ana",
43         "Apellido1": "Pérez",
44         "Apellido2": "Ruiz",
45         "DNI": "67890123F"

```

Ahora vamos a leer los datos de la tabla coches



GET <http://localhost:3000/api/leerdatoscoches> Send

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Body Cookies Headers (7) Test Results (1/1) 200 OK 1.26 s 9.4 KB Save Response

{ } JSON Preview Visualize

```

1  {
2      "success": true,
3      "message": "DATOS DE LA TABLA",
4      "data": [
5          {
6              "matricula": "1234ABC",
7              "marca": "Toyota",
8              "modelo": null,
9              "caballos": "120",
10             "persona_id": "1"
11          },
12          {
13              "matricula": "2345BCD",
14              "marca": "Renault",
15              "modelo": null,
16              "caballos": "90",
17              "persona_id": "1"
18          },
19          {
20              "matricula": "3456CDE",
21              "marca": "Ford",
22              "modelo": null

```

