

Parcial Bases de Datos

Dylan Felipe Avellaneda Garcia

Id:862216

Ing. William Alexander Matallana Porras

Corporación Universitaria Minuto de Dios

ingeniería de Sistemas

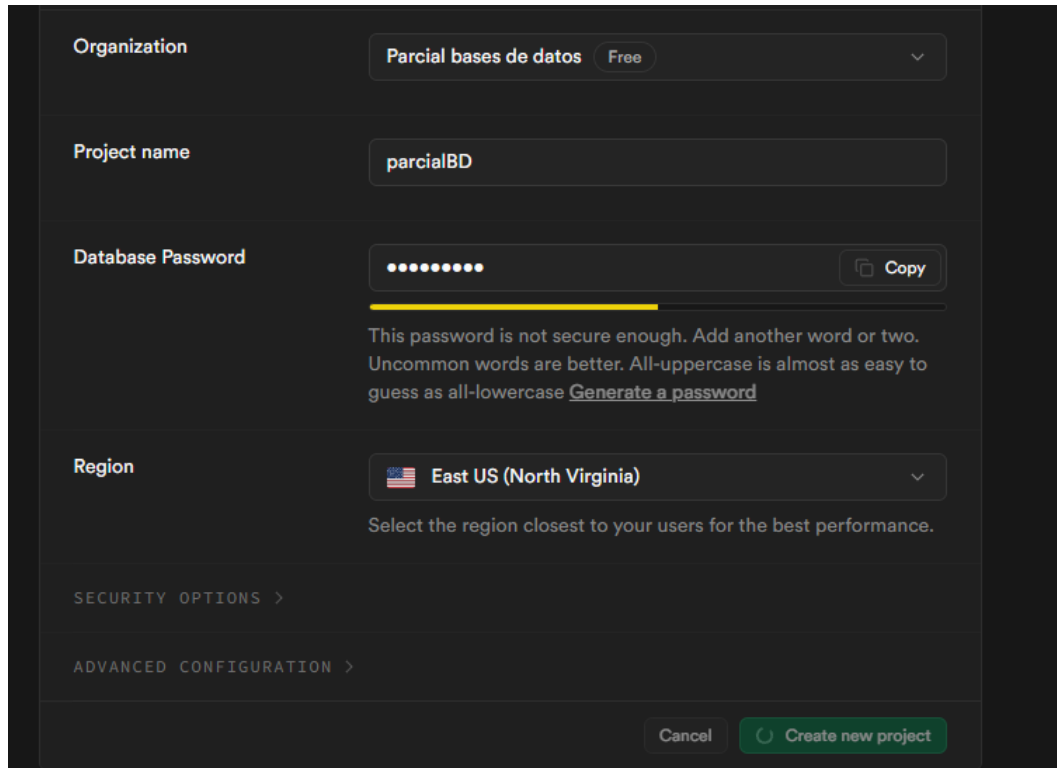
Bases de Datos Masivas

NRC:60747

Zipaquirá, 23 de abril de 2025

Parcial Bases de Datos

- Primero creamos una organización en supabase junto con la base de datos



The image shows the Supabase project creation interface. It includes fields for Organization (Parcial bases de datos), Project name (parcialBD), Database Password (masked with dots), and Region (East US (North Virginia)). A security warning is displayed below the password field. At the bottom, there are buttons for 'Cancel' and 'Create new project'.

Organization: Parcial bases de datos (Free)

Project name: parcialBD

Database Password: [masked] [Copy](#)

This password is not secure enough. Add another word or two. Uncommon words are better. All-uppercase is almost as easy to guess as all-lowercase. [Generate a password](#)

Region: East US (North Virginia)

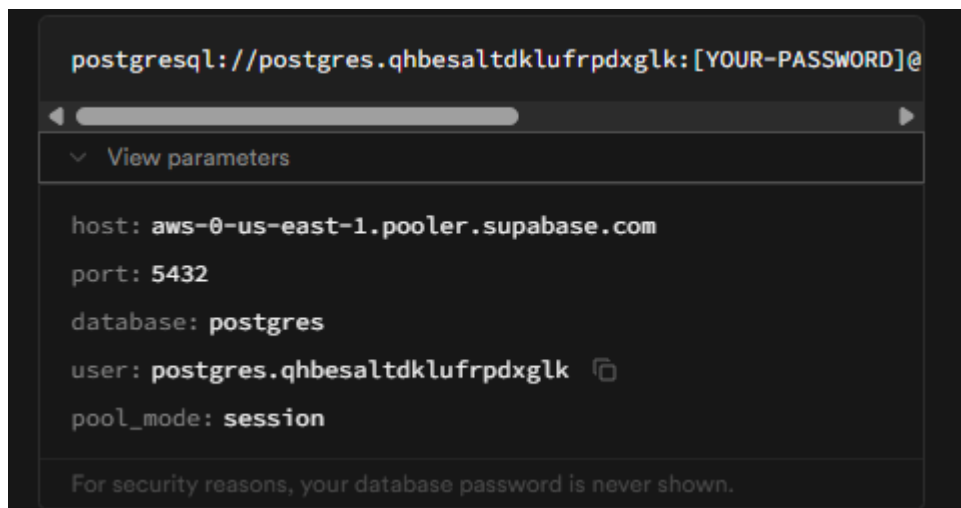
Select the region closest to your users for the best performance.

SECURITY OPTIONS >

ADVANCED CONFIGURATION >

Cancel Create new project

- Nos dirigimos al apartado de conexión en supabase para ver los parámetros de conexión



The image shows the Supabase connection parameters. It displays the PostgreSQL connection string and a list of parameters: host, port, database, user, and pool_mode. A note at the bottom states that the database password is never shown for security reasons.

postgresql://postgres.qhbesaltdklufrpdxgk:[YOUR-PASSWORD]@

View parameters

host: aws-0-us-east-1.pooler.supabase.com

port: 5432

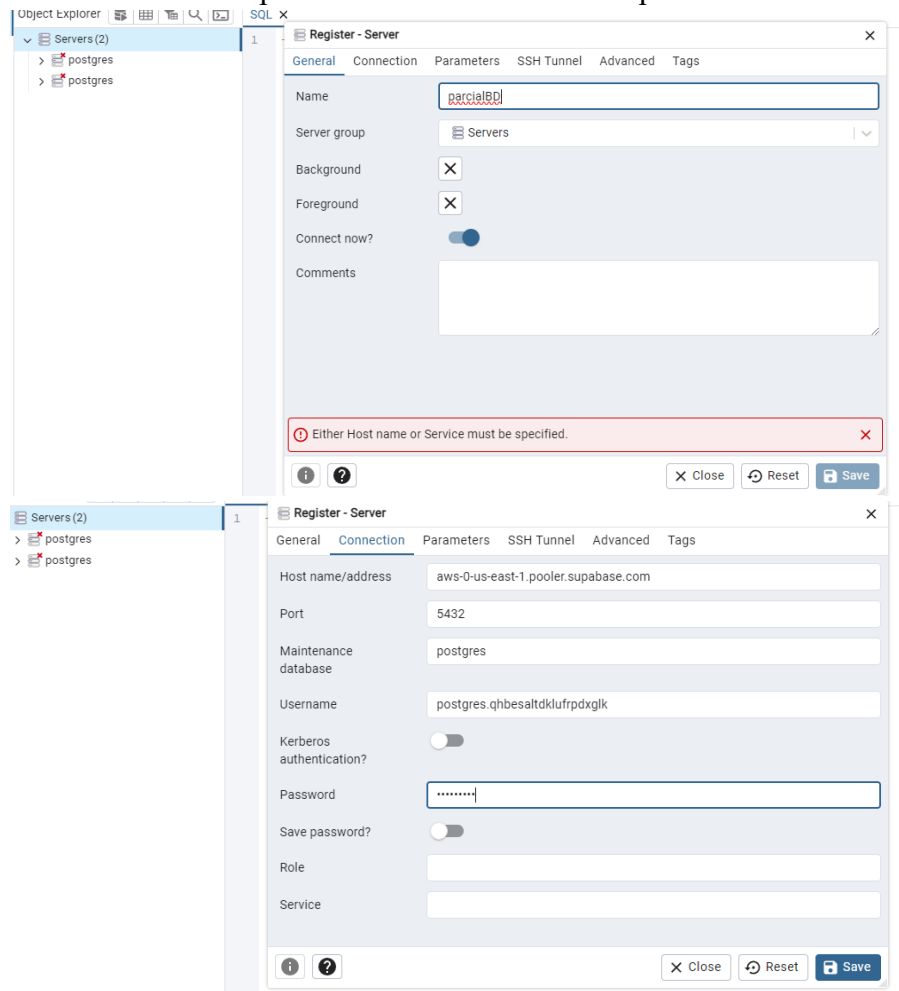
database: postgres

user: postgres.qhbesaltdklufrpdxgk

pool_mode: session

For security reasons, your database password is never shown.

- Abrimos PGAdmin para hacer la conexión con supabase



- Con la conexión ya realizada, nos dirigimos a supabase para la creación de las tablas de nuestra base de datos

1. Tabla restaurante:

```
1 CREATE TABLE Restaurante (  
2     id_rest INT PRIMARY KEY,  
3     nombre VARCHAR(100) NOT NULL,  
4     ciudad VARCHAR(100),  
5     direccion VARCHAR(150),  
6     fecha_apertura DATE  
7 );
```

2. Tabla empleado:

```
1 CREATE TABLE Empleado (  
2     id_empleado INT PRIMARY KEY,  
3     nombre VARCHAR(100) NOT NULL,  
4     rol VARCHAR(50),  
5     id_rest INT,  
6     CONSTRAINT fk_empleado_rest FOREIGN KEY (id_rest) REFERENCES Restaurante(id_rest)  
7 );
```

3. Tabla producto:

```
1 CREATE TABLE Producto (  
2     id_prod INT PRIMARY KEY,  
3     nombre VARCHAR(100) NOT NULL,  
4     precio NUMERIC(10,2) NOT NULL  
5 );
```

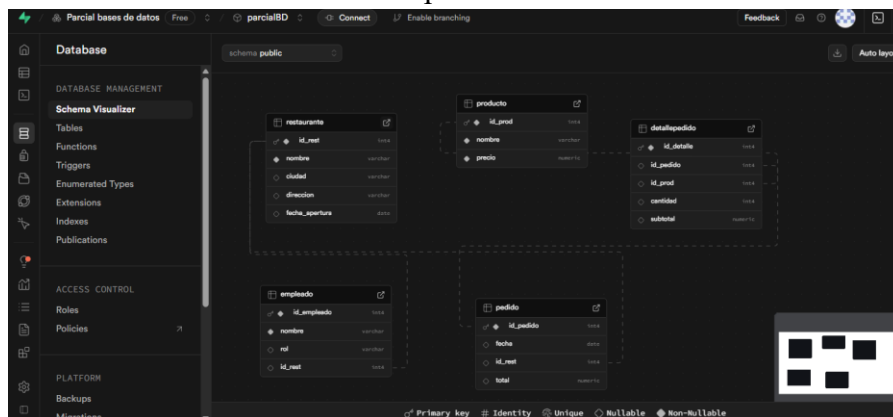
4. Tabla pedido:

```
1 CREATE TABLE Pedido (  
2     id_pedido INT PRIMARY KEY,  
3     fecha DATE,  
4     id_rest INT,  
5     total NUMERIC(10,2),  
6     CONSTRAINT fk_pedido_rest FOREIGN KEY (id_rest) REFERENCES Restaurante(id_rest)  
7 );  
8
```

5. Tabla detallepedido:

```
1 CREATE TABLE DetallePedido (  
2     id_detalle INT PRIMARY KEY,  
3     id_pedido INT,  
4     id_prod INT,  
5     cantidad INT,  
6     subtotal NUMERIC(10,2),  
7     CONSTRAINT fk_detalle_pedido FOREIGN KEY (id_pedido) REFERENCES Pedido(id_pedido),  
8     CONSTRAINT fk_detalle_producto FOREIGN KEY (id_prod) REFERENCES Producto(id_prod)  
9 );  
10
```

- Verificamos el modelo dentro supabase



- Ingresamos 50 registros en cada tabla.

1. Tabla restaurante:

```
INSERT INTO Restaurante (id_rest, nombre, direccion, ciudad) VALUES
(1, 'Restaurante Sol', 'Calle 1 #223', 'Bogota'),
(2, 'El Buen Sabor', 'Carrera 45 #98', 'Medellin'),
(3, 'Delicias del Norte', 'Av 5 #21-90', 'Cali'),
(4, 'Sabores de Casa', 'Calle 8 #60-70', 'Barranquilla'),
(5, 'La Parrilla', 'Cra 7 #12-34', 'Cartagena'),
(6, 'Comidas Rapidas Max', 'Calle 3 #23-45', 'Bogota'),
(7, 'El Rincón Paísa', 'Carrera 10 #50-60', 'Medellin'),
(8, 'Arepas & Sazon', 'Av 9 #100-10', 'Cali'),
(9, 'Bucandas Caseros', 'Calle 12 #44-88', 'Barranquilla'),
(10, 'Antojitos Express', 'Cra 15 #30-50', 'Cartagena'),
(11, 'Gusto Urbano', 'Calle 33 #12-11', 'Bogota'),
(12, 'La Cocina de Mama', 'Cra 8 #90-11', 'Medellin'),
(13, 'Fritanga Real', 'Av 10 #44-77', 'Cali'),
(14, 'El Fajon', 'Calle 20 #15-25', 'Barranquilla'),
(15, 'Donde Paco', 'Cra 10 #22-32', 'Cartagena'),
(16, 'Comidas del Alma', 'Calle 28 #77-66', 'Bogota'),
(17, 'La Sazon del Valle', 'Cra 2 #33-55', 'Medellin'),
(18, 'Rico y Liso', 'Av 6 #60-70', 'Cali'),
(19, 'El Sabor Criollo', 'Calle 18 #11-90', 'Barranquilla'),
(20, 'La Fonda', 'Cra 1 #44-23', 'Cartagena'),
(21, 'Casa de Adovea', 'Calle 10 #20-10', 'Bogota'),
(22, 'Donde Pepe', 'Cra 6 #66-66', 'Medellin'),
(23, 'El Buen Comer', 'Av 3 #30-30', 'Cali'),
(24, 'La Tienda del Sabor', 'Calle 4 #22-22', 'Barranquilla'),
(25, 'La Cocina Píspis', 'Cra 5 #00-00', 'Cartagena'),
(26, 'Comidas Calientes', 'Calle 14 #14-14', 'Bogota'),
(27, 'Sabor Costeño', 'Cra 12 #12-12', 'Medellin'),
(28, 'El Buen Plato', 'Av 10 #10-10', 'Cali'),
(29, 'Restaurante Colonial', 'Calle 7 #77', 'Barranquilla'),
(30, 'Comidas y Mas', 'Cra 5 #5-5', 'Cartagena'),
(31, 'Sabor y Aroma', 'Calle 11 #11-11', 'Bogota'),
```

2. Tabla empleado:

```
1 INSERT INTO empleado (id_empleado, nombre, rol, id_rest) VALUES
2 (1, 'Carlos Hernandez', 'Cocinero', 1),
3 (2, 'Luisa Fernandez', 'Mesero', 2),
4 (3, 'Ana Torres', 'Administrador', 3),
5 (4, 'Pedro Ramirez', 'Cajero', 4),
6 (5, 'Sofia Gomez', 'Mesera', 5),
7 (6, 'Jorge Lopez', 'Cocinero', 1),
8 (7, 'Daniela Ruiz', 'Cajero', 2),
9 (8, 'Marlene Salas', 'Administrador', 3),
10 (9, 'Andres Diaz', 'Mesero', 4),
11 (10, 'Lucia Vega', 'Cocinero', 5),
12 (11, 'Ivan Rios', 'Cajero', 1),
13 (12, 'Camila Mora', 'Mesera', 2),
14 (13, 'Esteban Suarez', 'Administrador', 3),
15 (14, 'Valeria Jimenez', 'Cocinero', 4),
16 (15, 'Santiago Paredes', 'Cajero', 5),
17 (16, 'Isabella Cruz', 'Mesera', 1),
18 (17, 'Martin Acosta', 'Cocinero', 2),
19 (18, 'Renata Herrera', 'Administrador', 3),
20 (19, 'Ismar Navarro', 'Mesero', 4),
21 (20, 'Alejandra Torres', 'Cajero', 5),
22 (21, 'Nicolas Bravo', 'Cocinero', 1),
23 (22, 'Ronica Pena', 'Administrador', 2),
24 (23, 'Emilia Linares', 'Mesera', 3),
25 (24, 'Gabriel Soto', 'Cocinero', 4),
26 (25, 'Daniel Torres', 'Cajero', 5),
27 (26, 'Fernanda Aguilas', 'Mesera', 1),
28 (27, 'Marina Castillo', 'Administrador', 2),
29 (28, 'Sara Molina', 'Cocinero', 3),
30 (29, 'Julian Duarte', 'Cajero', 4),
31 (30, 'Paula Romero', 'Mesera', 5),
32 (31, 'Sebastian Torres', 'Cocinero', 1),
```

3. Tabla producto:

```
1 INSERT INTO producto (id_prod, nombre, precio) VALUES
2 (1, 'Hamburguesa Clasica', 6.99),
3 (2, 'Pizze Margherita', 8.50),
4 (3, 'Taco de Pollo', 3.75),
5 (4, 'Ensalada Cesar', 5.25),
6 (5, 'Refresco Cola', 1.50),
7 (6, 'Agua Mineral', 1.00),
8 (7, 'Cerveza Artesanal', 4.20),
9 (8, 'Nachos con Queso', 4.99),
10 (9, 'Tee con Ejote', 2.00),
11 (10, 'Sopa del Dia', 3.50),
12 (11, 'Alitas BBQ', 6.00),
13 (12, 'Empanadas de Carne', 1.25),
14 (13, 'Papay Fritas', 2.50),
15 (14, 'Limonada Natural', 1.80),
16 (15, 'Cafe Americano', 1.70),
17 (16, 'Te Helado', 1.60),
18 (17, 'Sushi Roll Clasico', 7.90),
19 (18, 'Palln a la Plancha', 6.75),
20 (19, 'Costillas BBQ', 9.25),
21 (20, 'Pasta Alfredo', 7.10),
22 (21, 'Aves Fritas', 4.60),
23 (22, 'Burrito de Carne', 5.85),
24 (23, 'Cruquetos de Pollo', 3.15),
25 (24, 'Quesadilla Rusa', 4.40),
26 (25, 'Torta Mexicana', 5.40),
27 (26, 'Smoothie de Fresa', 2.90),
28 (27, 'Helado de Vainilla', 2.20),
29 (28, 'Milanesa de Res', 7.50),
30 (29, 'Sancocho', 6.50),
31 (30, 'Arroz con Queso', 2.40),
32 (31, 'Mojito Maracuya', 1.50),
```

4. Tabla pedido:

```
1 INSERT INTO pedido (id_pedido, fecha, id_rest, total) VALUES
2 (1, '2024-01-01', 1, 15.00),
3 (2, '2024-01-02', 2, 22.00),
4 (3, '2024-01-03', 3, 11.25),
5 (4, '2024-01-04', 1, 31.30),
6 (5, '2024-01-05', 2, 18.99),
7 (6, '2024-01-06', 4, 20.25),
8 (7, '2024-01-07', 3, 10.99),
9 (8, '2024-01-08', 1, 28.00),
10 (9, '2024-01-09', 2, 16.70),
11 (10, '2024-01-10', 3, 25.99),
12 (11, '2024-01-11', 4, 14.30),
13 (12, '2024-01-12', 1, 12.25),
14 (13, '2024-01-13', 2, 23.00),
15 (14, '2024-01-14', 3, 27.40),
16 (15, '2024-01-15', 4, 21.10),
17 (16, '2024-01-16', 1, 17.45),
18 (17, '2024-01-17', 2, 30.00),
19 (18, '2024-01-18', 3, 19.50),
20 (19, '2024-01-19', 4, 12.70),
21 (20, '2024-01-20', 1, 24.00),
22 (21, '2024-01-21', 2, 26.35),
23 (22, '2024-01-22', 3, 20.85),
24 (23, '2024-01-23', 4, 13.75),
25 (24, '2024-01-24', 1, 35.99),
26 (25, '2024-01-25', 2, 14.30),
27 (26, '2024-01-26', 3, 22.75),
28 (27, '2024-01-27', 4, 19.99),
29 (28, '2024-01-28', 1, 16.00),
30 (29, '2024-01-29', 2, 27.00),
31 (30, '2024-01-30', 3, 23.10),
32 (31, '2024-01-31', 4, 32.25);
```

Results Chart Export

Success. No rows returned

5. Tabla detallepedido:

```
1 INSERT INTO detallepedido (id_detalle, id_pedido, id_prod, cantidad, subtotal) VALUES
2 (1, 1, 1, 2, 15.00),
3 (2, 1, 2, 1, 3.50),
4 (3, 2, 2, 1, 8.50),
5 (4, 2, 19, 1, 9.25),
6 (5, 2, 5, 3, 4.50),
7 (6, 3, 4, 1, 5.25),
8 (7, 3, 5, 4, 6.00),
9 (8, 4, 19, 2, 18.50),
10 (9, 4, 13, 2, 5.00),
11 (10, 4, 14, 1, 1.00),
12 (11, 5, 17, 1, 7.00),
13 (12, 5, 18, 1, 6.75),
14 (13, 5, 5, 3, 4.50),
15 (14, 6, 20, 2, 14.20),
16 (15, 6, 6, 2, 2.00),
17 (16, 6, 5, 2, 3.00),
18 (17, 7, 25, 1, 5.40),
19 (18, 7, 10, 1, 3.50),
20 (19, 7, 2, 1, 8.50),
21 (20, 8, 1, 3, 20.97),
22 (21, 8, 13, 2, 5.00),
23 (22, 8, 5, 1, 3.50),
24 (23, 9, 24, 1, 4.00),
25 (24, 9, 22, 1, 9.85),
26 (25, 9, 6, 3, 3.00),
27 (26, 10, 28, 1, 7.30),
28 (27, 10, 7, 2, 8.00),
29 (28, 10, 9, 2, 4.00),
30 (29, 11, 30, 2, 4.00),
31 (30, 11, 6, 1, 1.00),
32 (31, 11, 11, 1, 6.00);
```

Results Chart Export

Success. No rows returned

- Ahora hacemos la conexión a la base de datos

```
JS baseDatos.js > [0] client
1 const { Client } = require('pg');
2
3 const client = new Client({
4   host: 'aws-0-us-east-1.pooler.supabase.com',
5   port: 5432,
6   user: 'postgres.qhbesaltdklufpdxgik',
7   password: 'lGME80s6ImCZvizc',
8   database: 'postgres',
9   ssl: { rejectUnauthorized: false } // necesario para Supabase
10 });
11
12 client.connect()
13   .then(() => console.log('Conectado a Supabase'))
14   .catch(err => console.error('Error en la conexión:', err.message));
15
16 module.exports = client;
```

- Verificamos que la conexión este funcionando correctamente

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Servidor corriendo en puerto 3000
Servidor corriendo en puerto 3000
Conectado a Supabase
```

- Organizamos las rutas de nuestro proyecto

```
✓ PARCIALBD  [🔍] [🔍] [🔄] [📄]

> node_modules
  routes
    JS detallePedido.js  U
    JS empleado.js      U
    JS pedido.js        U
    JS producto.js      U
    JS restaurante.js   U
    ~$rcial Bases de Datos.docx  U
    JS baseDatos.js     M
    JS index.js         M
    {} package-lock.json
    {} package.json
```

- Creamos las rutas de las apis para cada tabla

1. Tabla restaurante:

```
routes > restaurante.js
Click to add a breakpoint
1 require('express');
2 const express = require('express');
3 const client = require('../baseDatos');
4
5 // GET: Obtener todos los restaurantes
6 router.get('/', async (req, res) => {
7   try {
8     const result = await client.query('SELECT * FROM Restaurante');
9     res.json(result.rows);
10  } catch (error) {
11    res.status(500).json({ message: 'Error al obtener restaurantes', error: error.message });
12  }
13 });
14
15 // POST: Crear un restaurante
16 router.post('/', async (req, res) => {
17   const { id_rest, nombre, ciudad, direccion, fecha_apertura } = req.body;
18   try {
19     await client.query(
20       'INSERT INTO Restaurante (id_rest, nombre, ciudad, direccion, fecha_apertura) VALUES ($1, $2, $3, $4, $5)'
21     );
22     res.status(201).json({ message: 'Restaurante creado' });
23   } catch (error) {
24     res.status(500).json({ message: 'Error al crear restaurante', error: error.message });
25   }
26 });
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Servidor corriendo en puerto 3000
Servidor corriendo en puerto 3000
Conectado a Supabase
```

2. Tabla empleado:

```
routes > JS empleados > ...
1  const express = require('express');
2  const router = express.Router();
3  const client = require('../baseDatos');
4
5  router.get('/', async (req, res) => {
6    try {
7      const result = await client.query('SELECT * FROM Empleado');
8      res.json(result.rows);
9    } catch (error) {
10     res.status(500).json({ message: 'Error al obtener empleados', error: error.message });
11   }
12 });
13
14 router.post('/', async (req, res) => {
15   const { id_empleado, nombre, rol, id_rest } = req.body;
16   try {
17     await client.query(
18       'INSERT INTO Empleado (id_empleado, nombre, rol, id_rest) VALUES ($1, $2, $3, $4)',
19       [id_empleado, nombre, rol, id_rest]
20     );
21     res.status(201).json({ message: 'Empleado creado' });
22   } catch (error) {
23     res.status(500).json({ message: 'Error al crear empleado', error: error.message });
24   }
25 });
```

3. Tabla producto:

```
routes > JS productos > ...
1  const express = require('express');
2  const router = express.Router();
3  const client = require('../baseDatos');
4
5  router.get('/', async (req, res) => {
6    try {
7      const result = await client.query('SELECT * FROM Producto');
8      res.json(result.rows);
9    } catch (error) {
10     res.status(500).json({ message: 'Error al obtener productos', error: error.message });
11   }
12 });
13
14 router.post('/', async (req, res) => {
15   const { id_prod, nombre, precio } = req.body;
16   try {
17     await client.query(
18       'INSERT INTO Producto (id_prod, nombre, precio) VALUES ($1, $2, $3)',
19       [id_prod, nombre, precio]
20     );
21     res.status(201).json({ message: 'Producto creado' });
22   } catch (error) {
23     res.status(500).json({ message: 'Error al crear producto', error: error.message });
24   }
25 });
26
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

node

Servidor corriendo en puerto 3000
Servidor corriendo en puerto 3000
Conectado a Supabase

4. Tabla pedido:

```
routes > JS pedidos > ...
1  const express = require('express');
2  const router = express.Router();
3  const client = require('../baseDatos');
4
5  router.get('/', async (req, res) => {
6    try {
7      const result = await client.query('SELECT * FROM Pedido');
8      res.json(result.rows);
9    } catch (error) {
10     res.status(500).json({ message: 'Error al obtener pedidos', error: error.message });
11   }
12 });
13
14 router.post('/', async (req, res) => {
15   const { id_pedido, fecha, id_rest, total } = req.body;
16   try {
17     await client.query(
18       'INSERT INTO Pedido (id_pedido, fecha, id_rest, total) VALUES ($1, $2, $3, $4)',
19       [id_pedido, fecha, id_rest, total]
20     );
21     res.status(201).json({ message: 'Pedido creado' });
22   } catch (error) {
23     res.status(500).json({ message: 'Error al crear pedido', error: error.message });
24   }
25 });
26
```


5. Tabla detallpedido:

```

1 routes > JS detallePedidos.js > ...
2 const express = require('express');
3 const router = express.Router();
4 const client = require('../baseDatos');
5
6 router.get('/', async (req, res) => {
7   try {
8     const result = await client.query('SELECT * FROM DetallePedido');
9     res.json(result.rows);
10  } catch (error) {
11    res.status(500).json({ message: 'Error al obtener detalles', error: error.message });
12  }
13 });
14
15 router.post('/', async (req, res) => {
16   const { id_detalle, id_pedido, id_prod, cantidad, subtotal } = req.body;
17   try {
18     await client.query(
19       'INSERT INTO DetallePedido (id_detalle, id_pedido, id_prod, cantidad, subtotal) VALUES ($1, $2, $3, $4, $5)'
20     );
21     res.status(201).json({ message: 'Detalle creado' });
22   } catch (error) {
23     res.status(500).json({ message: 'Error al crear detalle', error: error.message });
24   }
25 });
26

```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

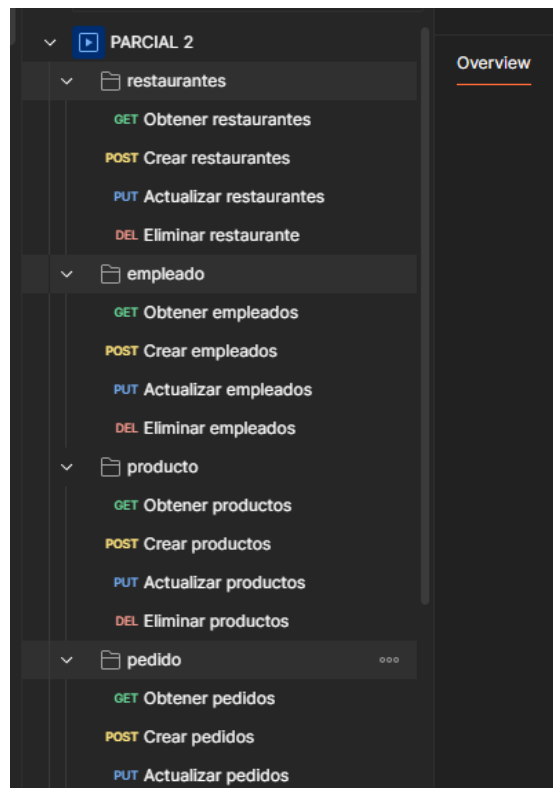
node

Servidor corriendo en puerto 3000

Servidor corriendo en puerto 3000

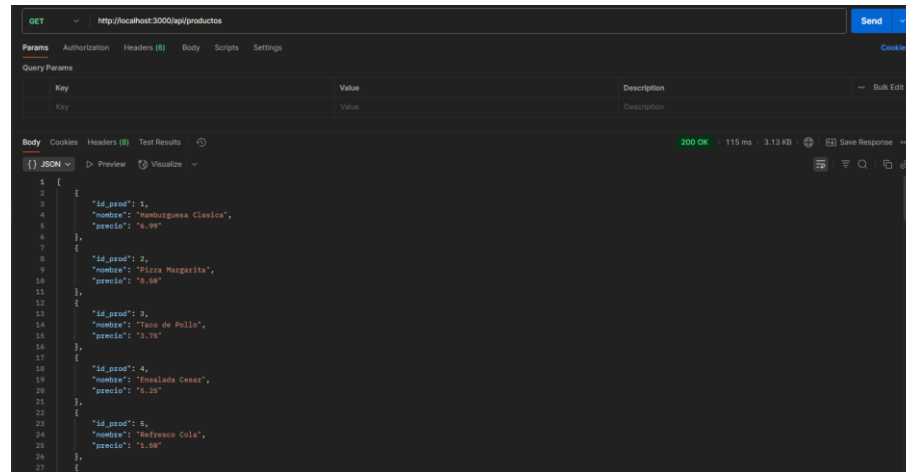
Conectado a Supabase

- Ahora ordenamos dentro de postman las peticiones correspondientes a cada tabla de una manera ordenada.

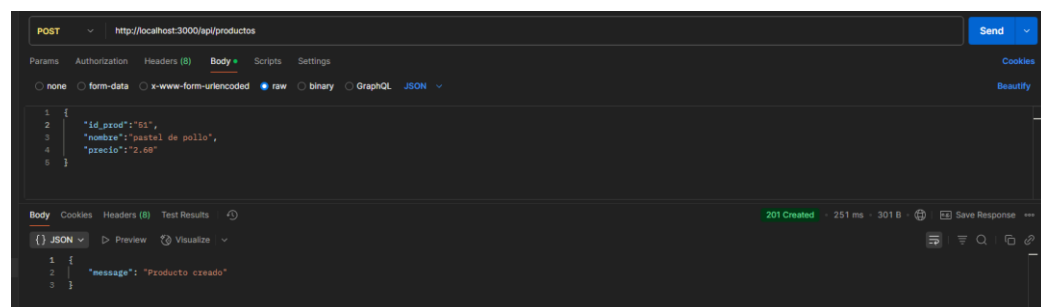


- Probamos las peticiones para ver si están funcionando correctamente, haremos la prueba con las peticiones de la tabla producto

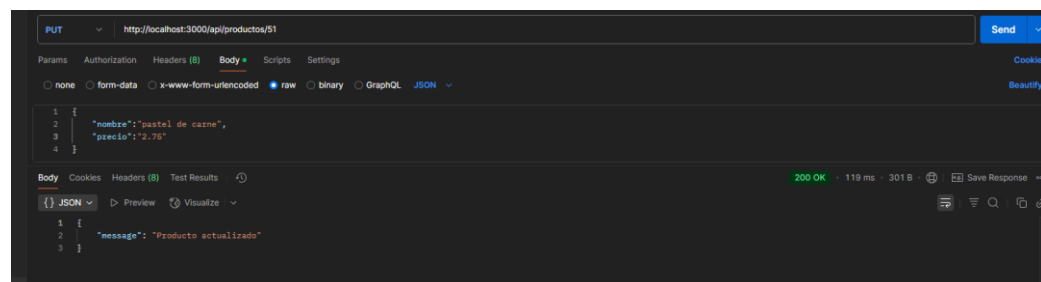
1. Get:



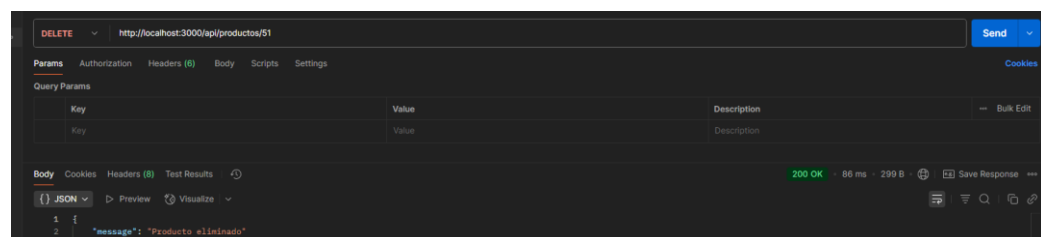
2. Post:



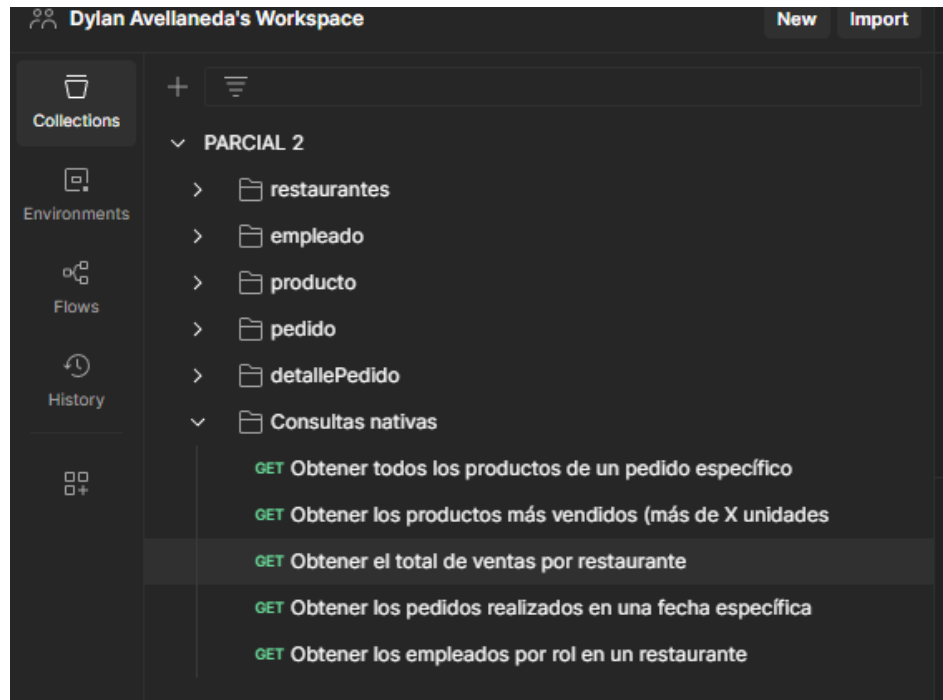
3. Put:



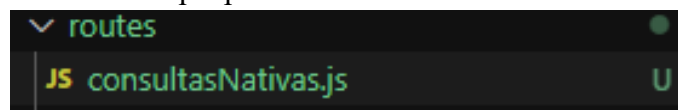
4. Delete:



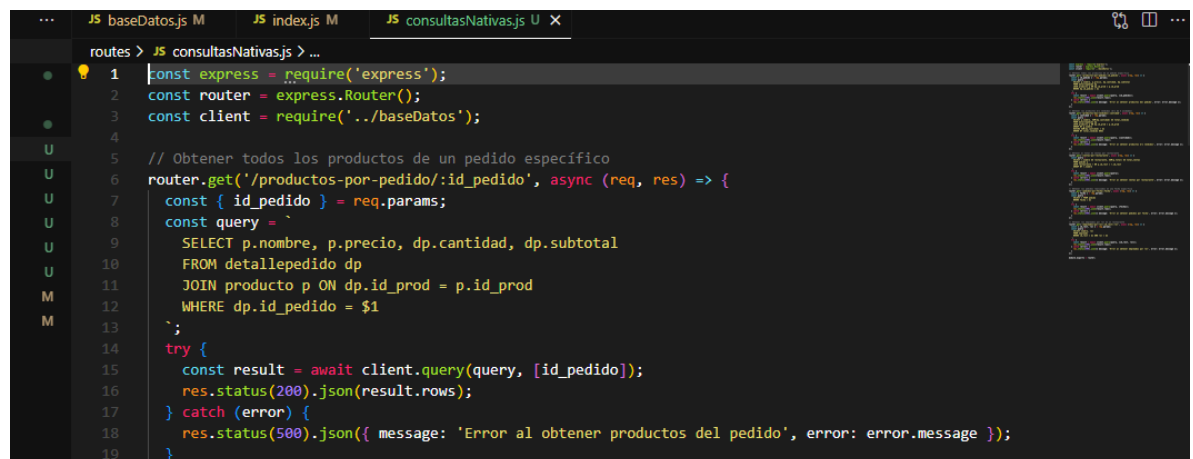
- Ahora en postman creamos una carpeta para las consultas nativas que vamos a realizar dentro de la base de datos



- De igual forma creamos un archivo dentro de nuestra carpeta de routes para la creación de apis para las consultas nativas



- Hacemos la conexión de nuestro archivo de consultas nativas con el index y la base de datos e iniciamos la creación de las apis para cada consulta nativa. Luego hacemos la prueba de la petición en postman para verificar que este funcionando de manera correcta.

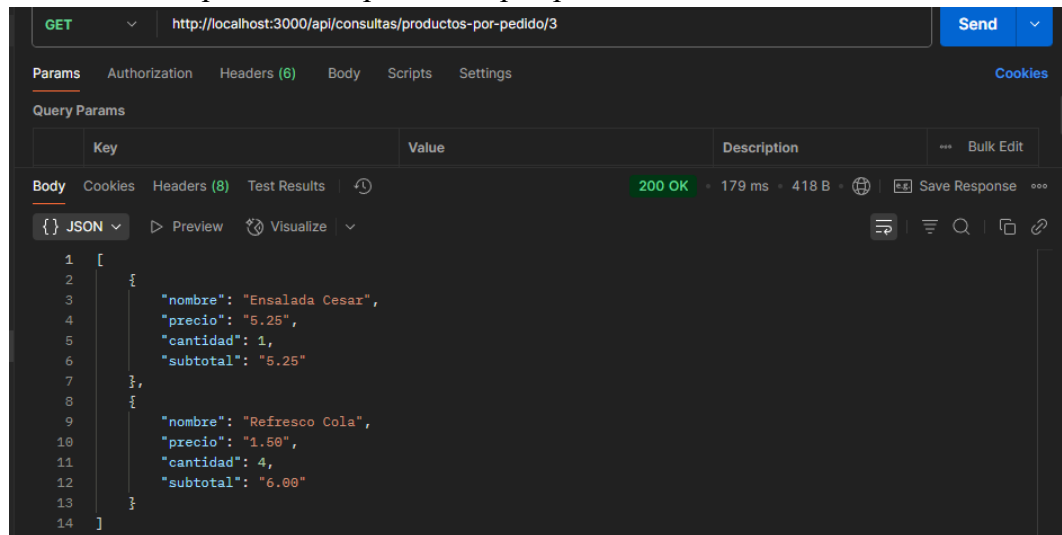


1. Obtener todos los productos de un pedido específico:

- Esta consulta utiliza un INNER JOIN entre las tablas DetallePedido y Producto para obtener el nombre, precio y cantidad de cada producto asociado a un pedido determinado. Es útil para conocer en detalle qué productos se incluyeron en un pedido.

```
5 // Obtener todos los productos de un pedido específico
6 router.get('/productos-por-pedido/:id_pedido', async (req, res) => {
7   const { id_pedido } = req.params;
8   const query = `
9     SELECT p.nombre, p.precio, dp.cantidad, dp.subtotal
10    FROM detallepedido dp
11    JOIN producto p ON dp.id_prod = p.id_prod
12    WHERE dp.id_pedido = $1
13  `;
14   try {
15     const result = await client.query(query, [id_pedido]);
16     res.status(200).json(result.rows);
17   } catch (error) {
18     res.status(500).json({ message: 'Error al obtener productos del pedido', error: error.message });
19   }
20 });
```

- <http://localhost:3000/api/consultas/productos-por-pedido/3>



The screenshot shows a web browser with the URL `http://localhost:3000/api/consultas/productos-por-pedido/3` in the address bar. The browser's developer tools are open, showing the response body in JSON format. The response is a 200 OK status with a response time of 179 ms and a size of 418 B. The JSON data is as follows:

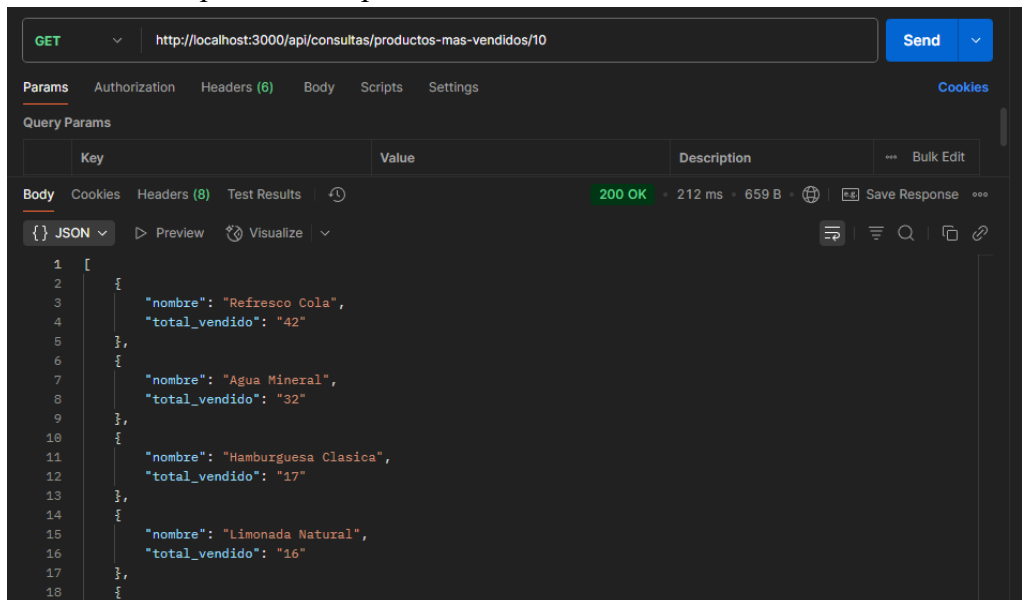
```
[
  {
    "nombre": "Ensalada Cesar",
    "precio": "5.25",
    "cantidad": 1,
    "subtotal": "5.25"
  },
  {
    "nombre": "Refresco Cola",
    "precio": "1.50",
    "cantidad": 4,
    "subtotal": "6.00"
  }
]
```

2. Obtener los productos más vendidos (más de X unidades):

- Se hace un INNER JOIN entre DetallePedido y Producto para contar cuántas unidades de cada producto se han vendido. Luego, se agrupa por el nombre del producto y se filtra usando HAVING para mostrar solo los que superan cierta cantidad. Esta consulta permite identificar los productos más populares.

```
22 // Obtener los productos más vendidos (más de X unidades)
23 router.get('/productos-mas-vendidos/:cantidad', async (req, res) => {
24   const { cantidad } = req.params;
25   const query = `
26     SELECT p.nombre, SUM(dp.cantidad) AS total_vendido
27     FROM detallepedido dp
28     JOIN producto p ON dp.id_prod = p.id_prod
29     GROUP BY p.nombre
30     HAVING SUM(dp.cantidad) > $1
31     ORDER BY total_vendido DESC
32   `;
33   try {
34     const result = await client.query(query, [cantidad]);
35     res.status(200).json(result.rows);
36   } catch (error) {
37     res.status(500).json({ message: 'Error al obtener productos más vendidos', error: error.message });
38   }
39 });
```

- <http://localhost:3000/api/consultas/productos-mas-vendidos/10>



The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:3000/api/consultas/productos-mas-vendidos/10
- Status:** 200 OK
- Response Time:** 212 ms
- Response Size:** 659 B
- Body:** JSON array of products and their total sales.

| Key | Value | Description |
|---------------|---------------------|-------------|
| nombre | Refresco Cola | |
| total_vendido | 42 | |
| nombre | Agua Mineral | |
| total_vendido | 32 | |
| nombre | Hamburguesa Clasica | |
| total_vendido | 17 | |
| nombre | Limonada Natural | |
| total_vendido | 16 | |

3. Obtener el total de ventas por restaurante:

- Mediante un INNER JOIN entre Pedido y Restaurante, se suman los totales de todos los pedidos realizados en cada restaurante. Se agrupa por el nombre del restaurante para conocer cuánto ha vendido cada uno en total.

```

41 // Obtener el total de ventas por restaurante
42 router.get('/ventas-por-restaurante', async (req, res) => {
43   const query = `
44     SELECT r.nombre AS restaurante, SUM(p.total) AS total_ventas
45     FROM pedido p
46     JOIN restaurante r ON p.id_rest = r.id_rest
47     GROUP BY r.nombre
48   `;
49   try {
50     const result = await client.query(query);
51     res.status(200).json(result.rows);
52   } catch (error) {
53     res.status(500).json({ message: 'Error al obtener ventas por restaurante', error: error.message });
54   }
55 });

```

- <http://localhost:3000/api/consultas/ventas-por-restaurante>

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** <http://localhost:3000/api/consultas/ventas-por-restaurante>
- Status:** 200 OK
- Response Body (JSON):**

```

[
  {
    "restaurante": "Delicias del Norte",
    "total_ventas": "300.83"
  },
  {
    "restaurante": "Sabores de Casa",
    "total_ventas": "225.04"
  },
  {
    "restaurante": "Restaurante Sol",
    "total_ventas": "295.23"
  },
  {
    "restaurante": "El Buen Sabor",
    "total_ventas": "334.68"
  }
]

```

4. Obtener los pedidos realizados en una fecha específica:

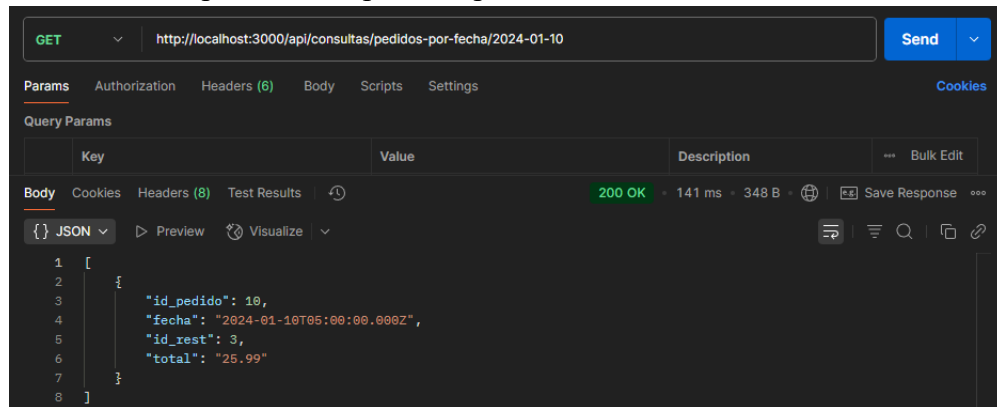
- Se realiza un INNER JOIN entre Pedido y Restaurante para listar los pedidos que se realizaron en una fecha específica, mostrando también de qué restaurante provienen. Esto sirve para llevar un control de pedidos diarios.

```

// Obtener los pedidos realizados en una fecha específica
router.get('/pedidos-por-fecha/:fecha', async (req, res) => {
  const { fecha } = req.params;
  const query = `
    SELECT * FROM pedido
    WHERE fecha = $1
  `;
  try {
    const result = await client.query(query, [fecha]);
    res.status(200).json(result.rows);
  } catch (error) {
    res.status(500).json({ message: 'Error al obtener pedidos por fecha', error: error.message });
  }
});

```

- <http://localhost:3000/api/consultas/pedidos-por-fecha/2024-01-10>

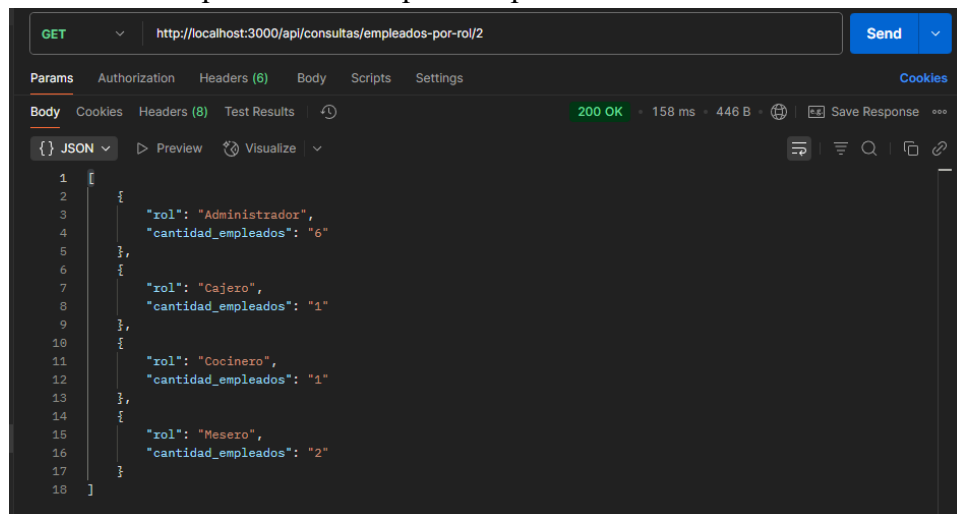


5. Obtener los empleados por rol en un restaurante:

- Esta consulta agrupa los empleados por su cargo (rol) dentro de un restaurante específico, usando GROUP BY sobre la columna del cargo. Permite conocer cuántos empleados cumplen con cada rol en un restaurante.

```
72 // Obtener los empleados por rol en un restaurante
73 router.get('/empleados-por-rol/:id_rest', async (req, res) => {
74   const { id_rest } = req.params;
75   try {
76     const result = await client.query(`
77       SELECT e.rol, COUNT(*) AS cantidad_empleados
78       FROM Empleado e
79       WHERE e.id_rest = $1
80       GROUP BY e.rol
81     `, [id_rest]);
82     res.json(result.rows);
83   } catch (error) {
84     res.status(500).json({ message: 'Error al obtener empleados por rol', error: error.message });
85   }
86 });
87
```

- <http://localhost:3000/api/consultas/empleados-por-rol/2>



NOTA: Las peticiones que se hacen dentro de postman siempre en con la url del local host junto con el puerto que establecimos para conexión y la ruta que asignamos para nuestras peticiones específicas que están dentro de nuestro archivo index, dependiendo de la petición que hagamos se cambia la ruta o se ponen condicionales para ejecutar la petición como en algunos casos cuando realiza la petición dependiendo del id.

```
// Importar las rutas
app.use('/api/restaurantes', require('./routes/restaurant'));
app.use('/api/empleados', require('./routes/empleados'));
app.use('/api/productos', require('./routes/producto'));
app.use('/api/pedidos', require('./routes/pedido'));
app.use('/api/detalles', require('./routes/detallePedido'));
app.use('/api/consultas', require('./routes/consultasNativas'));
```