Parcial 2

Paula Alejandra Martínez Huertas

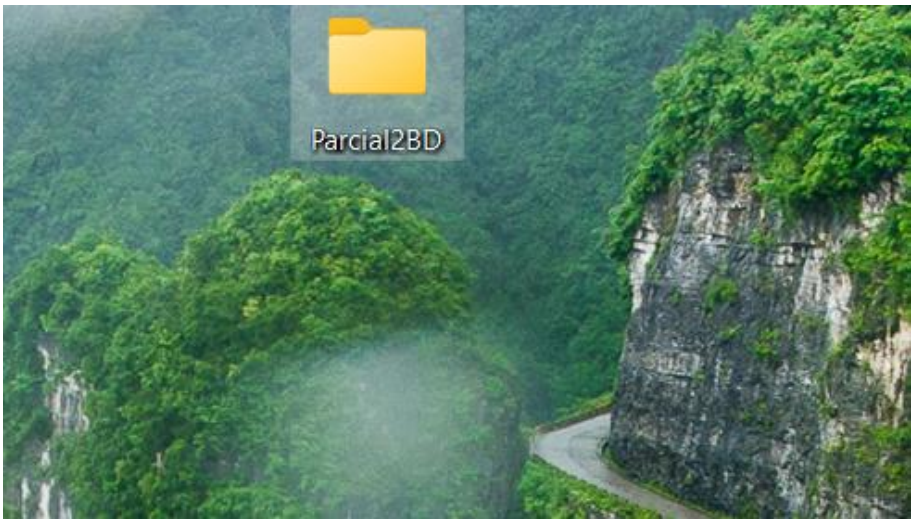
Ingeniería de sistemas, Corporación Universitaria Minuto de Dios

NRC 60747: Base de datos masivas

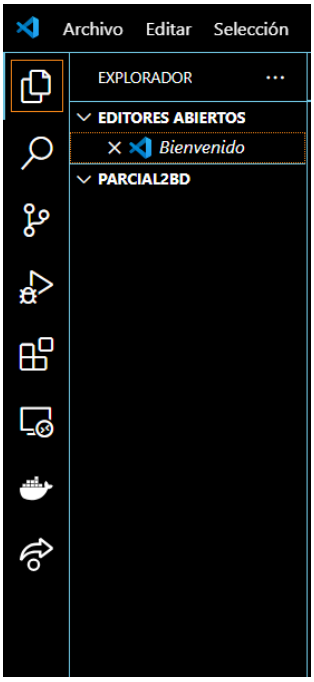
Ing. William Matallana Porras

Abril 25, 2025

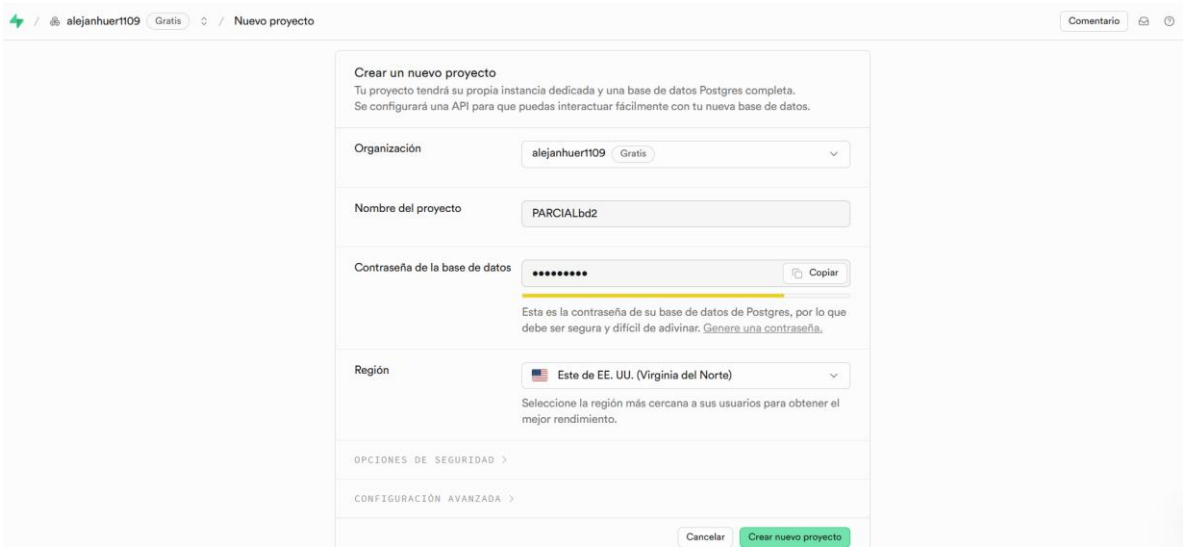
1. Creamos una carpeta en el escritorio, en este caso llamada ***“Parcial2BD”***.



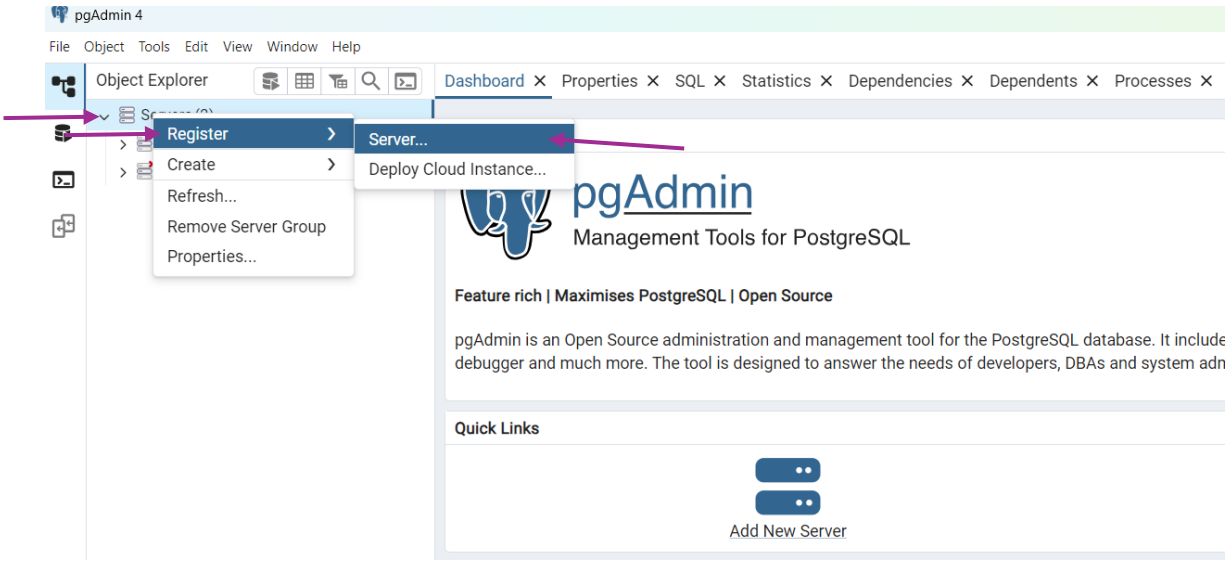
2. Abrimos la carpeta creada en el espacio de trabajo de nuestro Visual Studio Code.



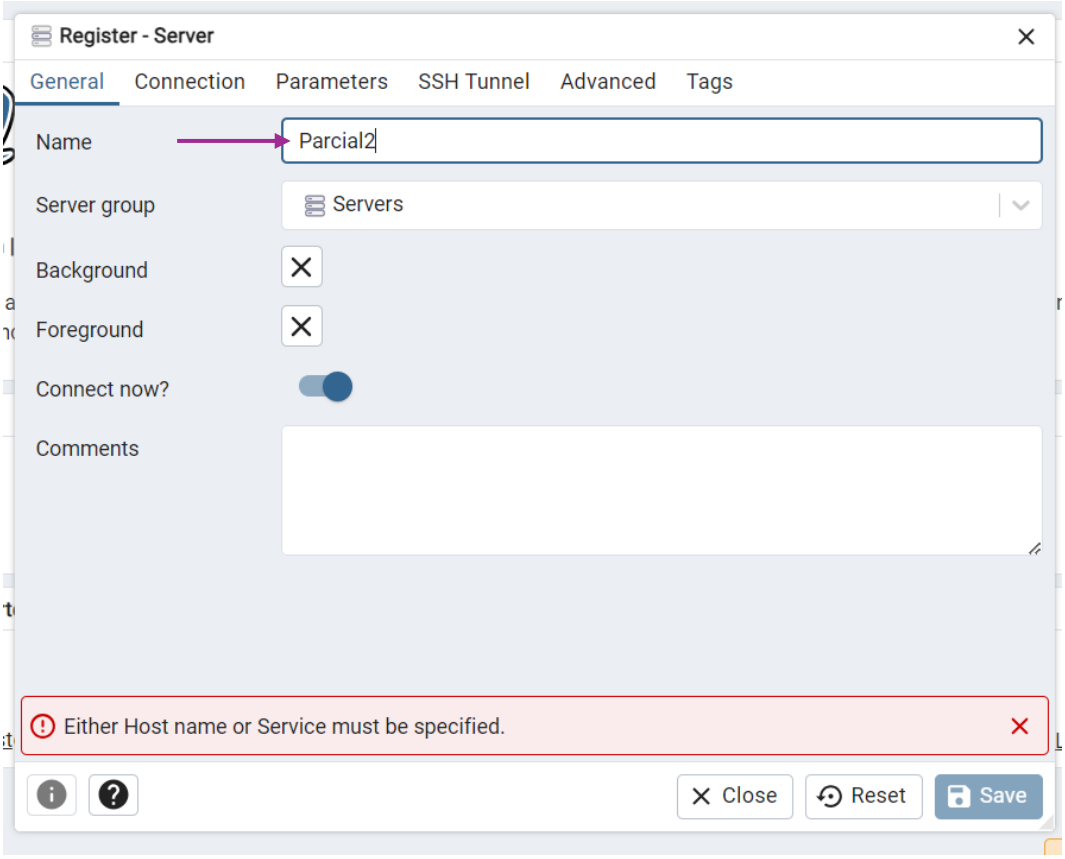
3. Creamos un nuevo proyecto en Supabase, en este caso llamado ***"PARCIALbd2"***, se establece la contraseña de la base de datos y se da clic en la opción Crear nuevo proyecto.



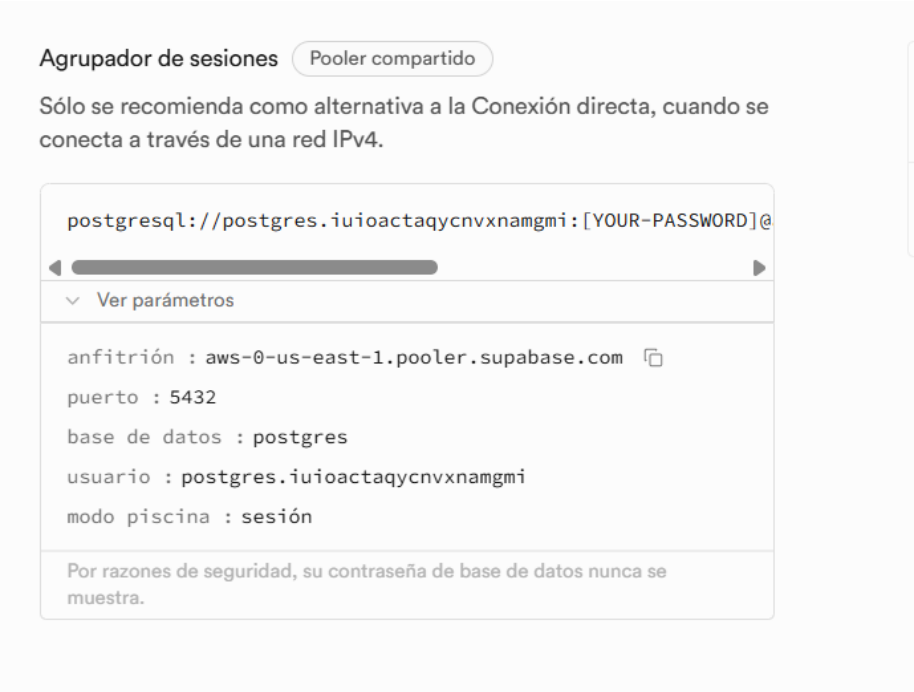
4. En pgAdmin se crea un nuevo servidor para establecer la conexión con Supabase. Para ello, dar clic derecho sobre ***Server*** en el menú lateral izquierdo, desplegar la opción ***Register*** y seleccionar la opción ***Server.***

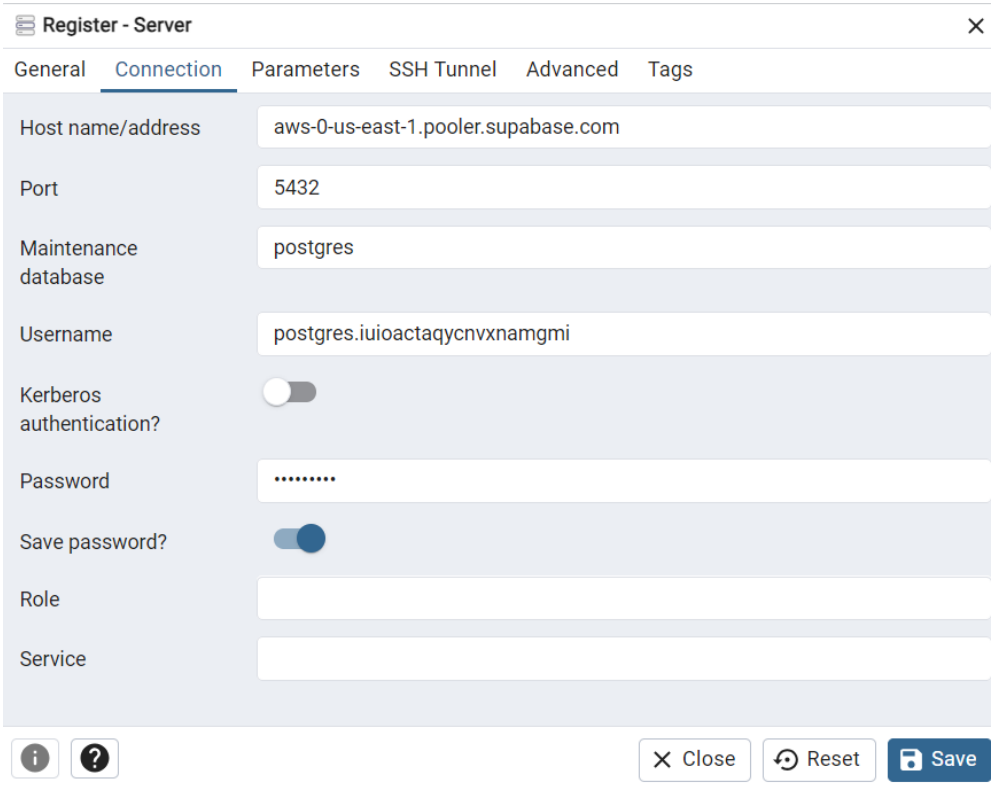


5. Nos aparece una ventana nueva, en la sección ***General*** en el campo ***Name*** se establece un nombre para el servidor.

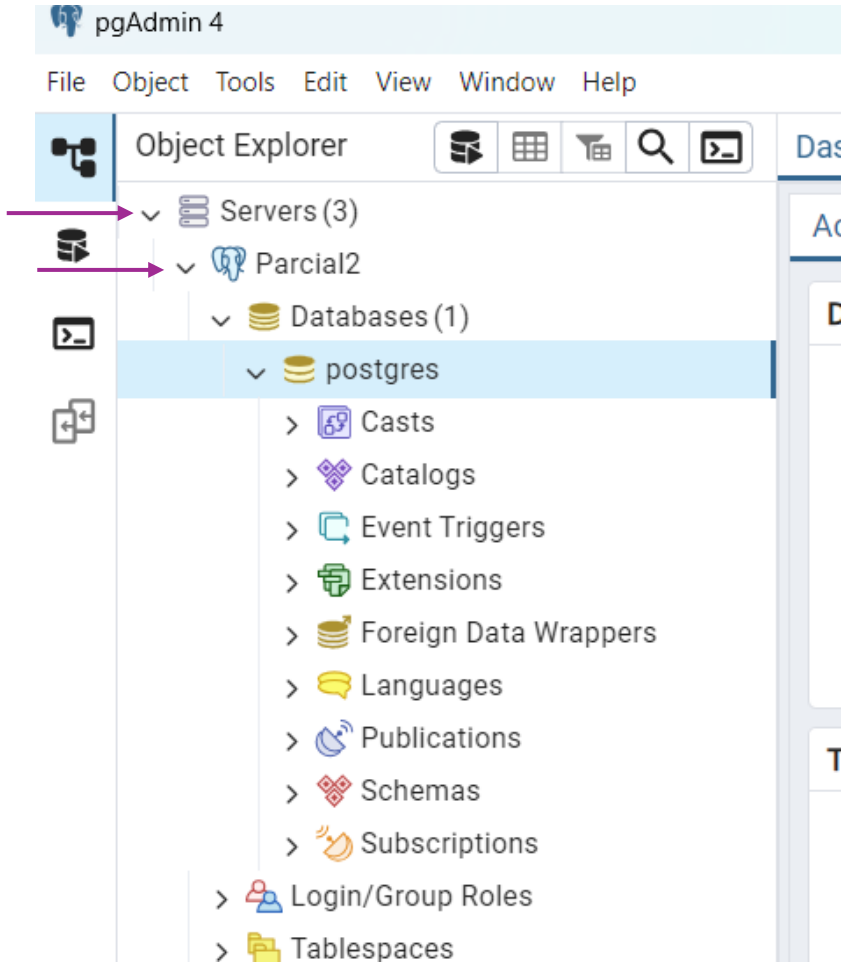


6. En el campo ***Connection*** se realiza la respectiva conexión tomando los datos de nuestro proyecto creado en Supabase.


Agrupador de sesiones
Pooler compartido
Sólo se recomienda como alternativa a la Conexión directa, cuando se conecta a través de una red IPv4.
postgresql://postgres.iuioactaqycnvxnamgmi:[YOUR-PASSWORD]@
Ver parámetros
anfitrión : aws-0-us-east-1.pooler.supabase.com
puerto : 5432
base de datos : postgres
usuario : postgres.iuioactaqycnvxnamgmi
modo piscina : sesión
Por razones de seguridad, su contraseña de base de datos nunca se muestra.


Register - Server
General
Connection
Parameters
SSH Tunnel
Advanced
Tags
Host name/address
aws-0-us-east-1.pooler.supabase.com
Port
5432
Maintenance database
postgres
Username
postgres.iuioactaqycnvxnamgmi
Kerberos authentication?
Password
Save password?
Role
Service
Close
Reset
Save

7. Guardar los cambios de la configuración de la conexión dando clic en el botón ***Save*** y debe estar creado nuestro servidor ***"Parcial2"*** al desplegar en el menú latera izquierdo el campo ***Servers***.



8. Creamos las tablas en nuestra base de datos localizada en el servidor ***Parcial2*** en pgAdmin, llamada por defecto ***postgres***.

Tabla Restaurante

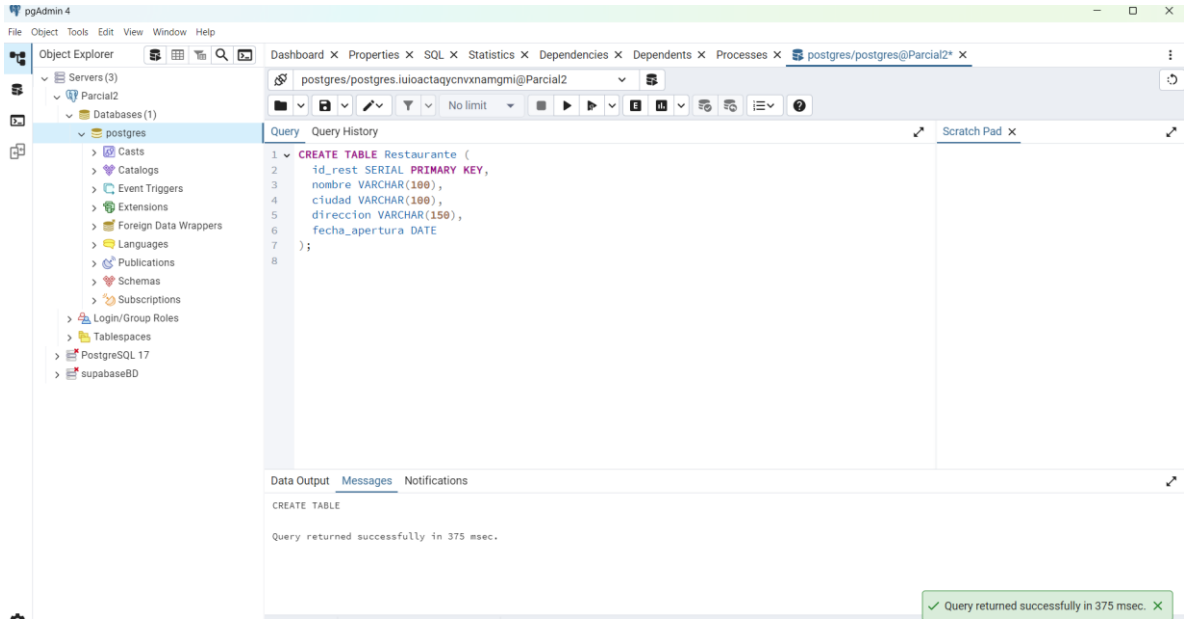


Tabla Empleado

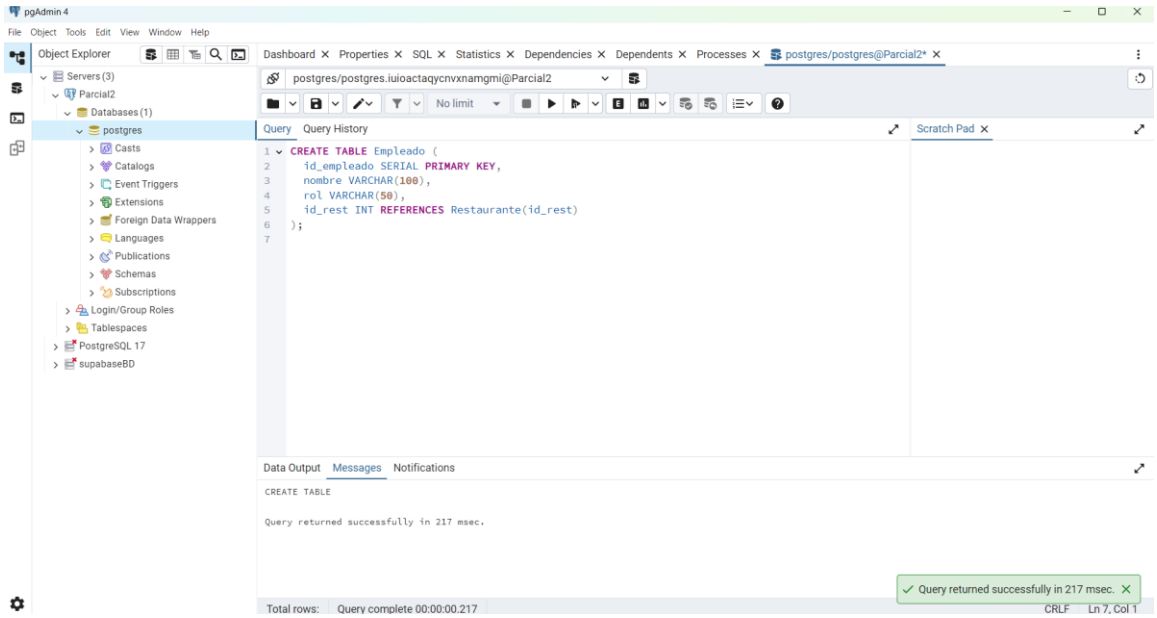


Tabla Producto

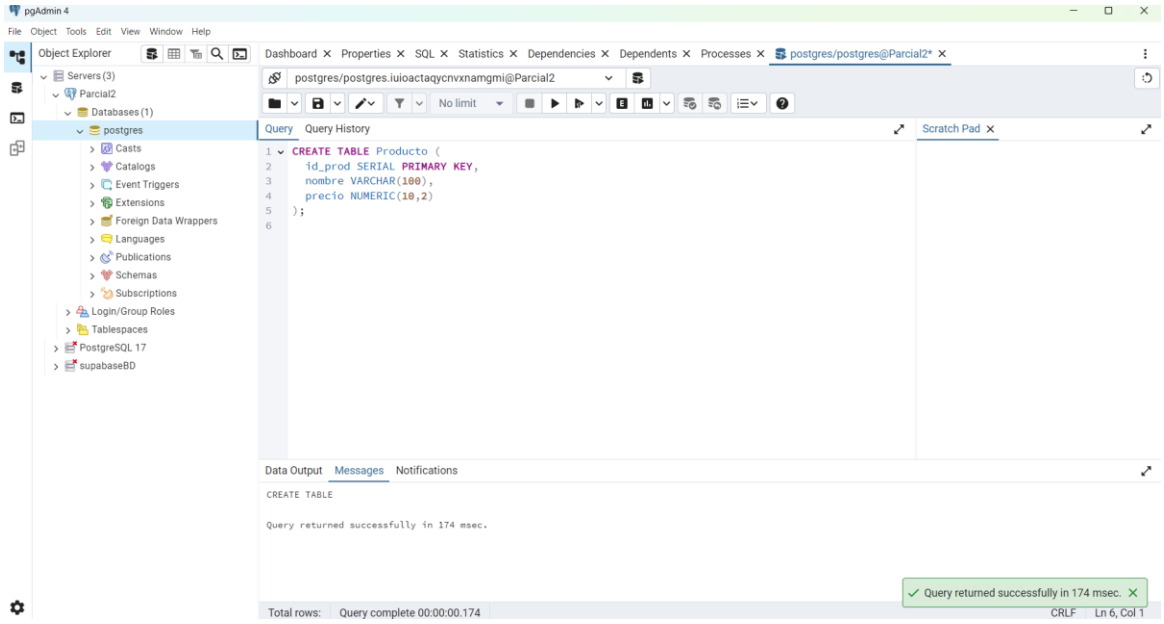


Tabla Pedido

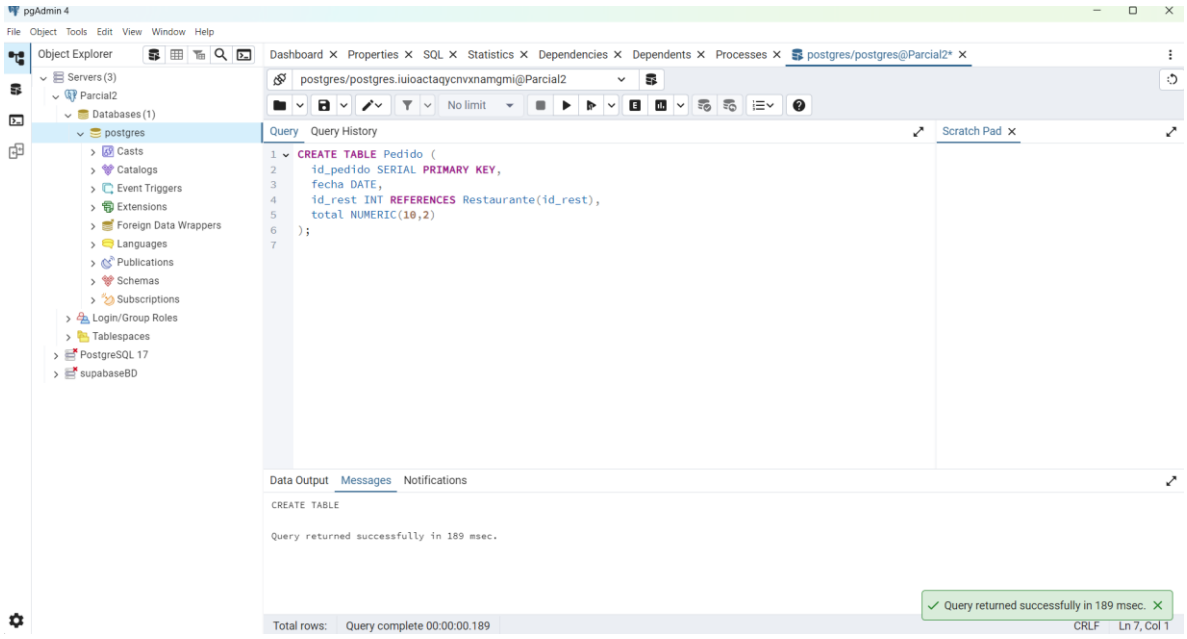
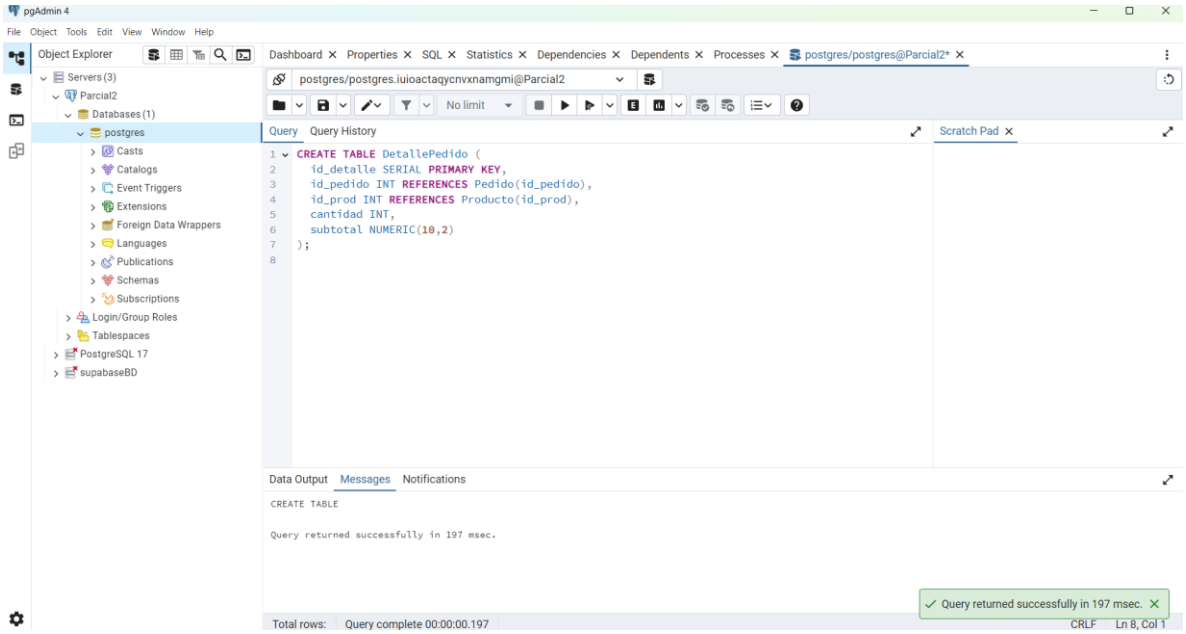
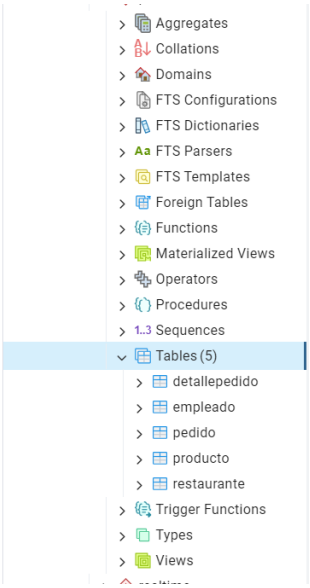
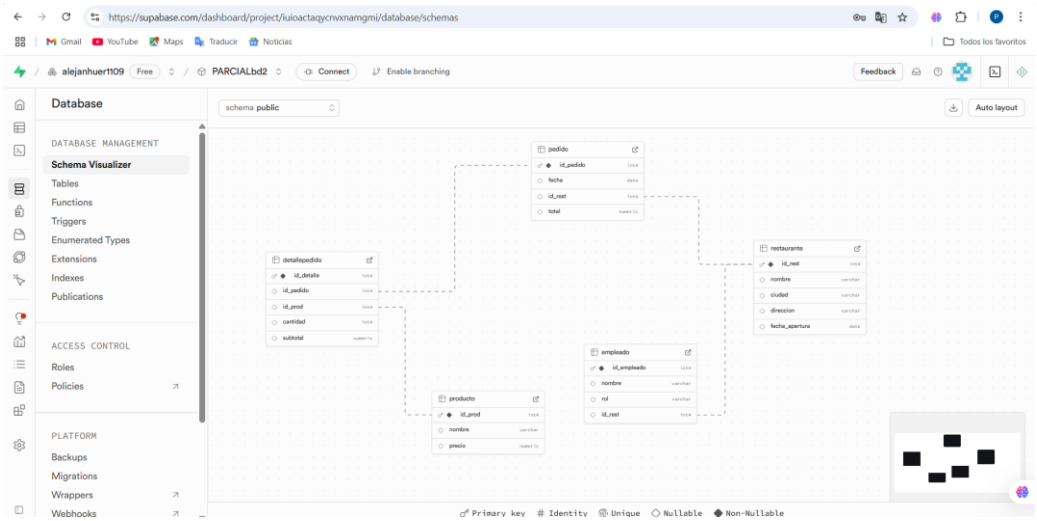


Tabla DetallePedido

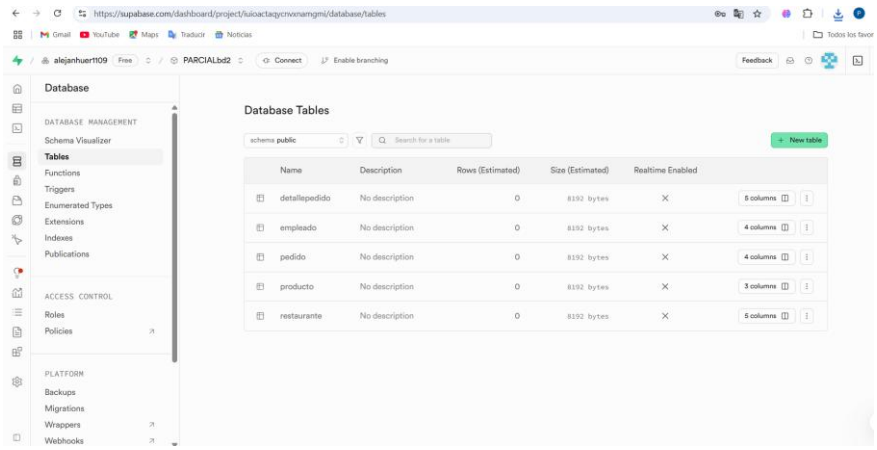




9. Modelo relacional obtenido desde Supabase.

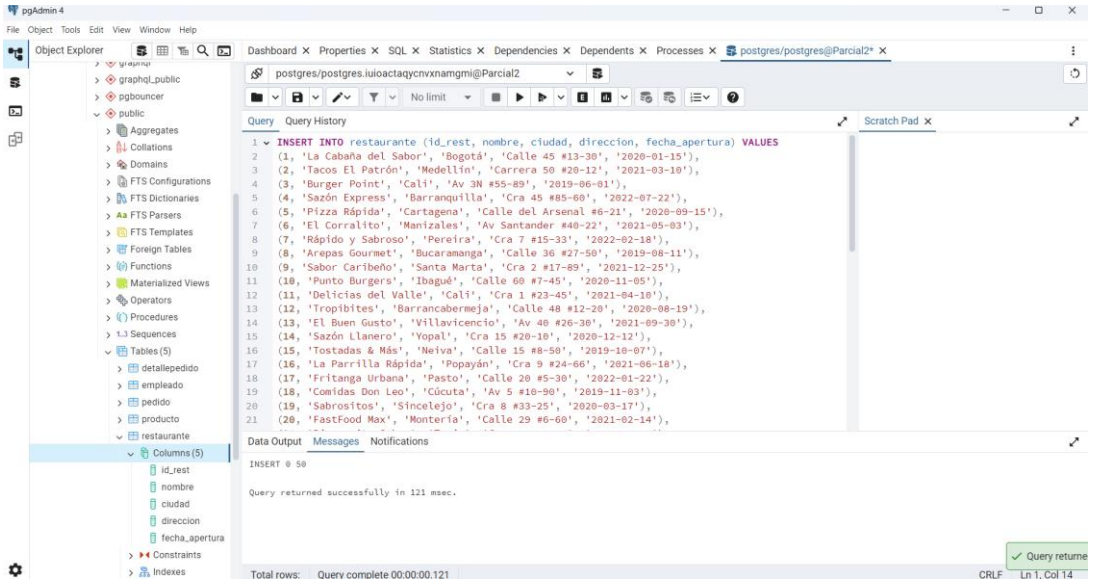


10. Verificamos que las tablas se hallan creado correctamente en el entorno de Supabase también.



11. Insertamos datos a cada tabla (50 registros por tabla) desde pgAdmin y validamos con la conexión en la interfaz de Supabase.

Tabla restaurante



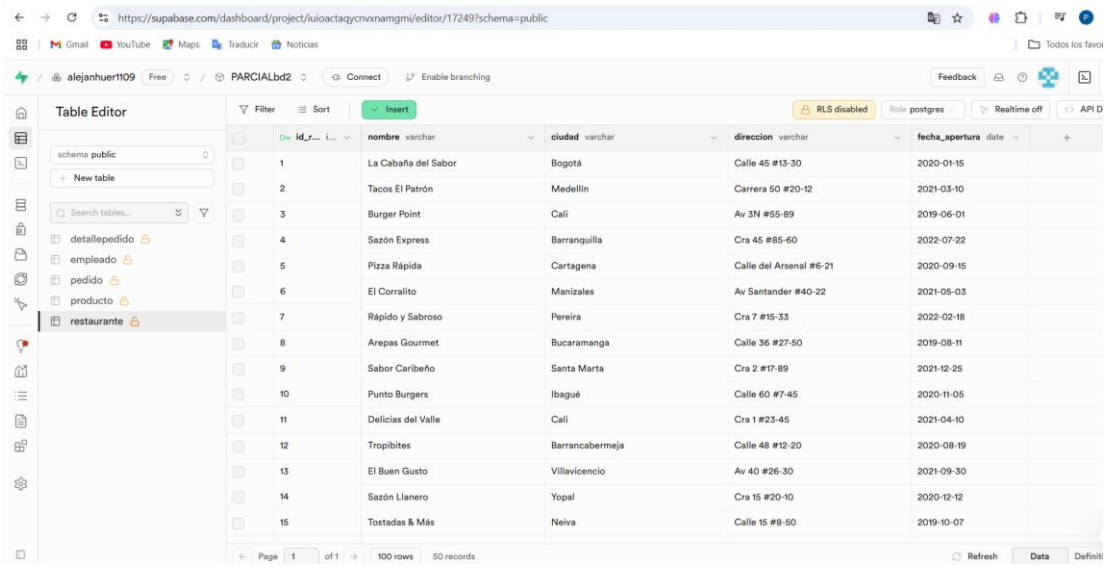
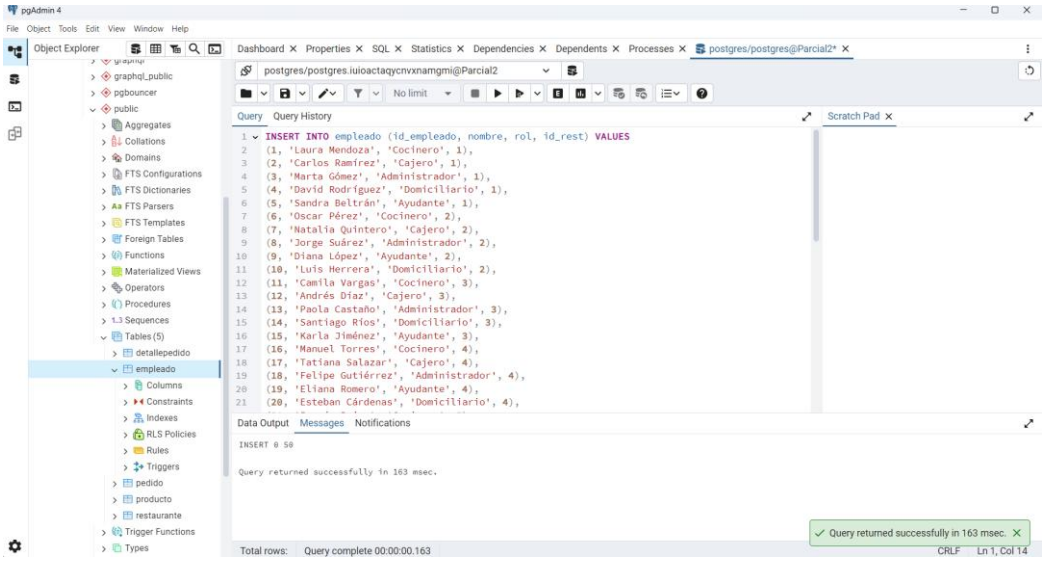


Tabla empleado



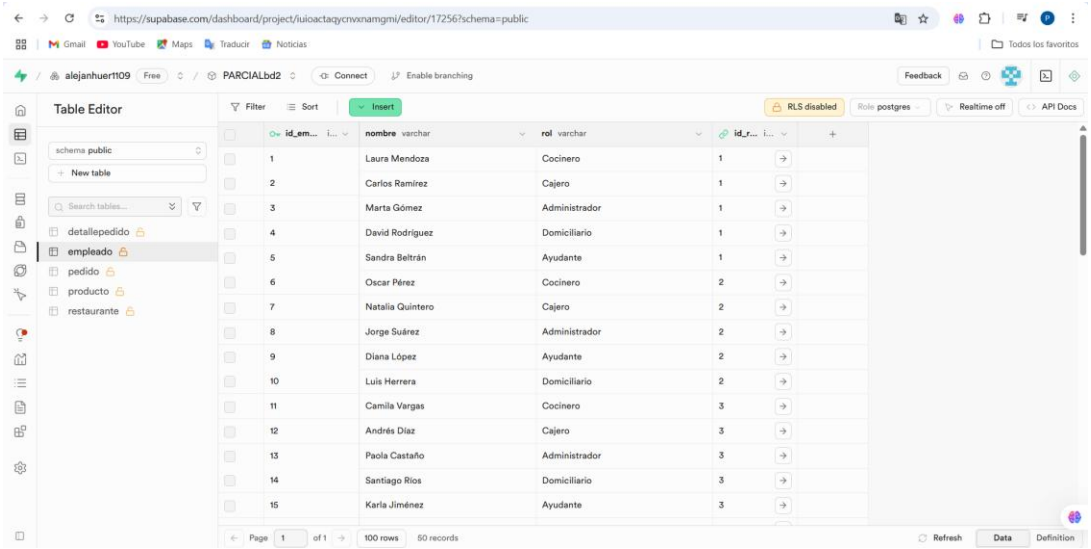
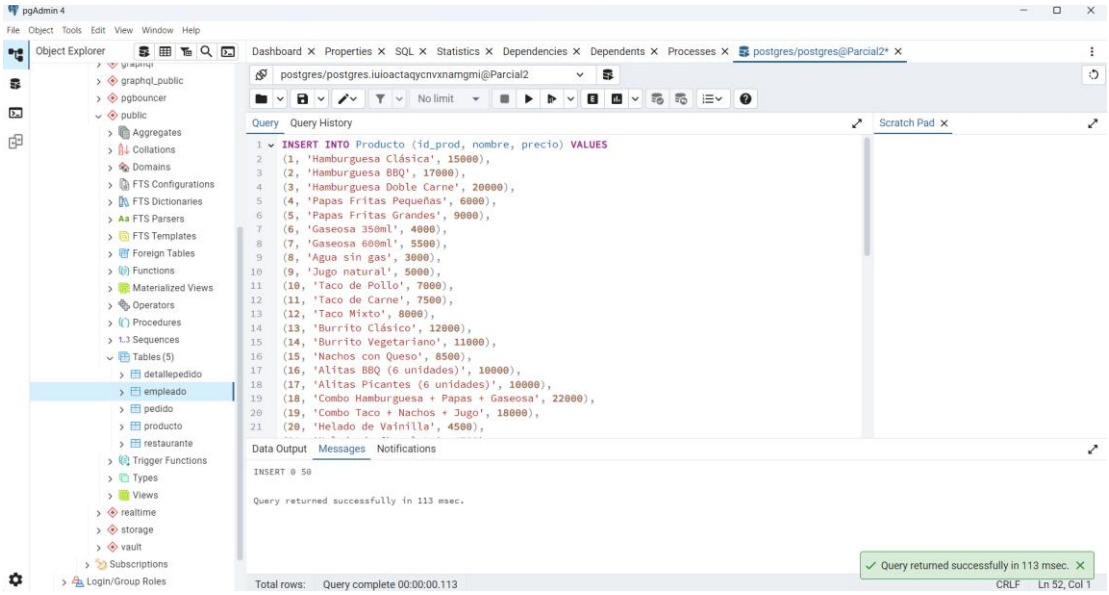


Tabla producto



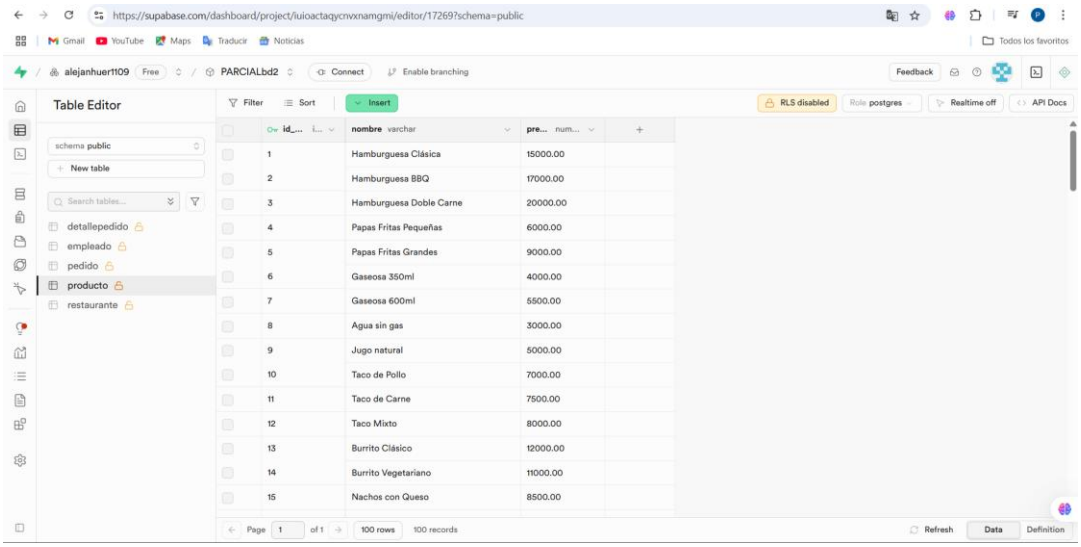
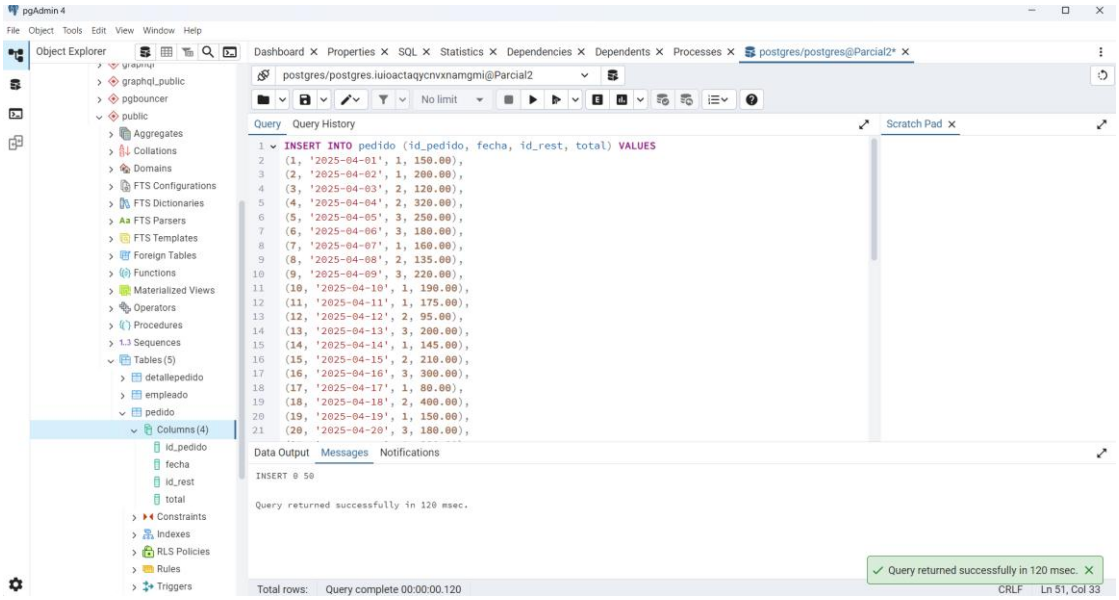


Tabla pedido



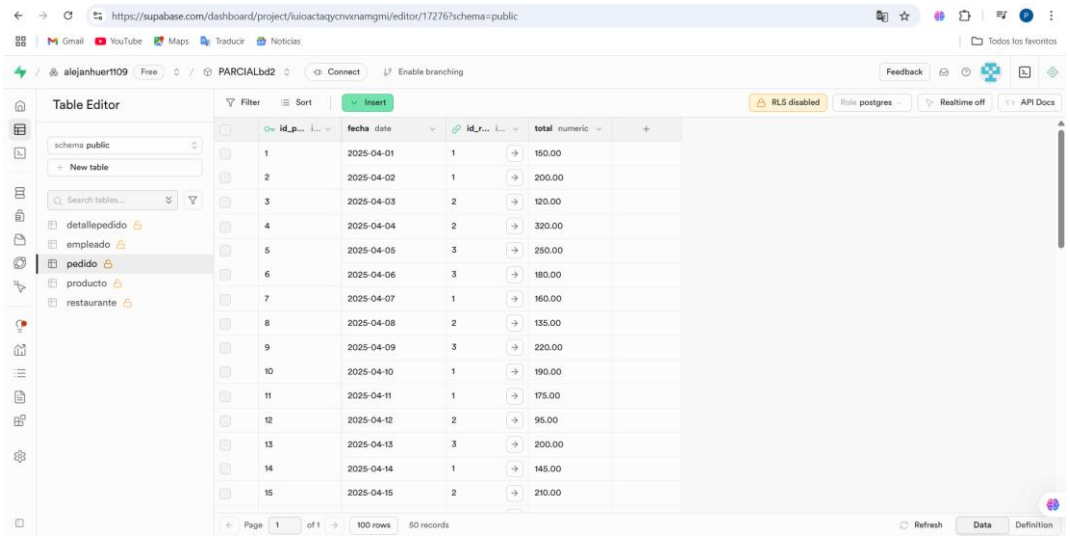
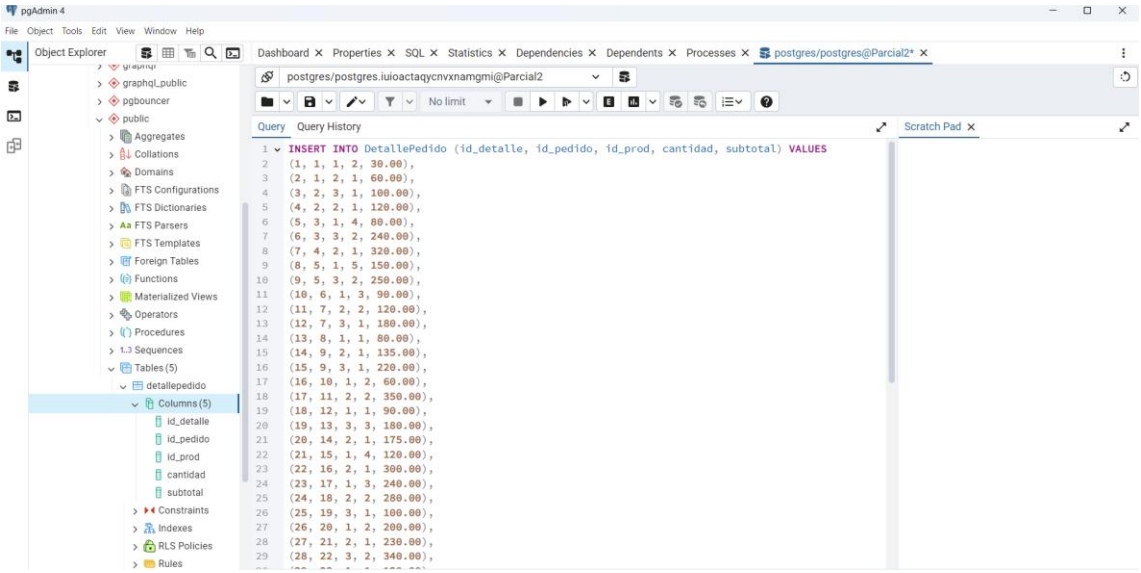
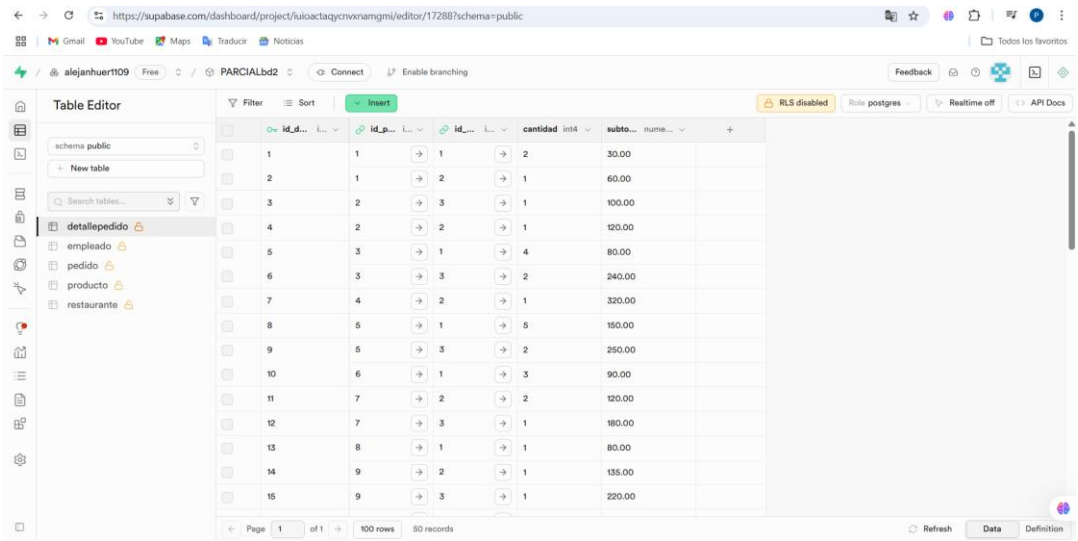
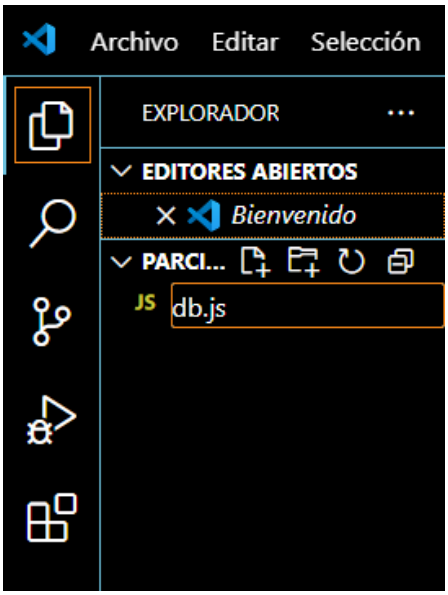


Tabla DetallePedido



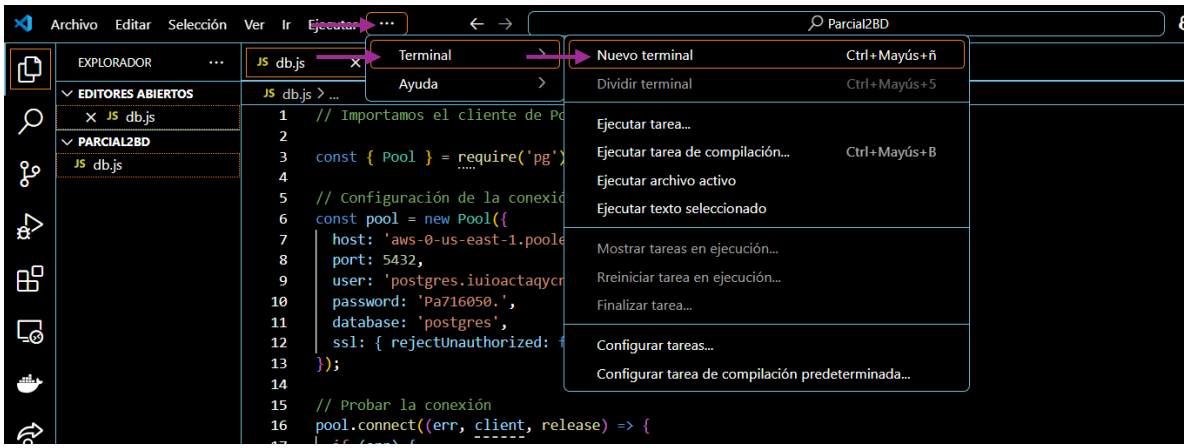


12. En el entorno de Visual Studio Code dentro de la carpeta a trabajar, en este caso ***"Parcial2BD"*** creamos el archivo en lenguaje JavaScript llamado ***"db.js"*** para ejercer la conexión con la base de datos con el archivo index.js, pgAdmin y Supabase.

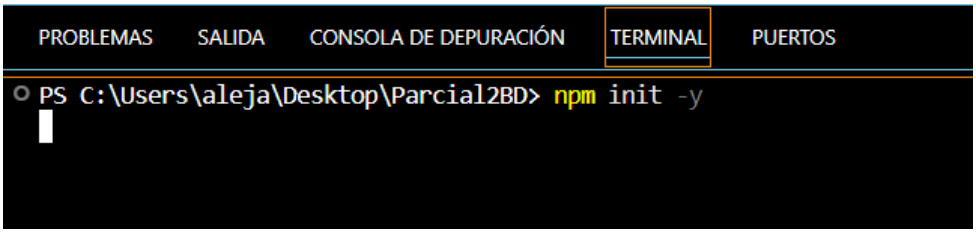


13. El código desarrollado en el archivo **db.js** establece una conexión con una base de datos PostgreSQL utilizando el cliente pg en Node.js. Se tienen en cuenta parámetros importantes de nuestra conexión con Supabase como: **host, port, user, password y database.**

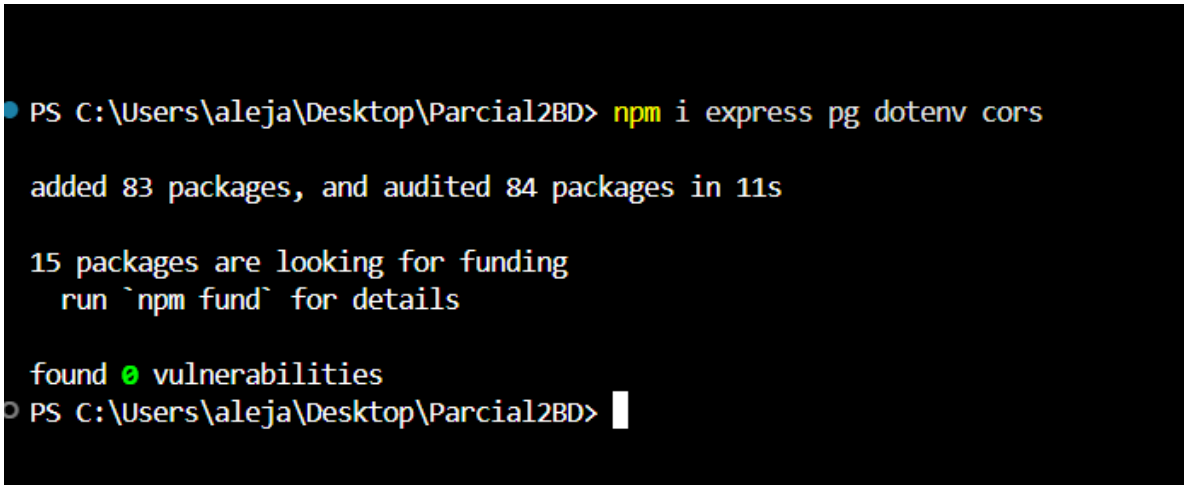
14. Abrimos la terminal dentro de nuestro proyecto.



15. Inicializamos un nuevo proyecto con el comando **npm init -y**, ya que sirve para crear rápidamente un archivo package.json con valores por defecto, sin tener que responder preguntas en la terminal.



16. Para instalar los paquetes de PostgreSQL usamos el comando **npm i express pg dotenv cors** que instala uno o más paquetes de Node.js en el proyecto.



17. Se utiliza **node - - watch index.js** en la terminal para que se detecten los cabios ejercidos en el proyecto automáticamente.

```
PS C:\Users\aleja\Desktop\Parcial2BD> node - - watch index.js
Welcome to Node.js v22.14.0.
Type ".help" for more information.
>
```

18. Dentro del archivo ***index.js*** se va a jalar la librería express y la conexión a la base de datos.

```
JS db.js    JS index.js U ×
JS index.js > ...
1   //Jalo la libreria express
2   const express = require('express');
3   //jalo la conexion a la base de datos
4   const connection = require('./db');
```

19. Se crea una instancia de la aplicación de Express. Esto es clave en cualquier aplicación que utilice este framework.

```
6
7   //aqui vamos a poder usar express por medio de esta variable app
8   const app = express();
9
```

20. Activamos el middleware que permite al servidor manejar cuerpos de solicitudes con formato JSON.

```
10
11    //Encargado de parsear todo alos json
12    app.use(express.json());
```

21. Se define una ruta en una aplicación Express para manejar solicitudes HTTP GET. Configura el middleware para que Express pueda interpretar las solicitudes con datos codificados en URL, luego responde a las solicitudes GET en esta ruta enviando un mensaje de

texto simple y por último responde a las solicitudes GET en esta ruta enviando una respuesta JSON con un mensaje, el puerto y un estado.

```
14  //Para que me lea las direcciones URL
15  app.use(express.urlencoded({extended:true}));
16
17  app.get('/api/prueba', (req, res)=>{
18      res.send('estoy respondiendo por la api')
19  });
20
21  app.get('/api/prueba2',(req, res)=>{
22      res.status(200).json({
23          message:'api funciona bien',
24          port:PORT,
25          status:'exitoso'
26
27      });
28
29  });
```

22. Este código establece el puerto de conexión para un servidor Express y lo pone en funcionamiento. Se establece una constante PORT con el valor 3000, que es el puerto en el que el servidor escuchará las solicitudes entrantes. Luego la función ***app.listen(PORT, () => { ... })*** inicia el servidor y lo hace escuchar en el puerto definido (3000). Si el servidor se inicia correctamente, se ejecuta la función de callback.

```
30  //Crear puerto de conexion del servidor
31  const PORT = 3000;
32
33  //La conexion la va a escuchar por el puerto 3000 y si
34  app.listen(PORT, ()=>{
35      console.log('El servidor esta corriendo');
36
37  });
38
```

23. Se crea el CRUD para la tabla ***Restaurante*** en nuestro archivo index.js.

Post

```
//CRUD RESTAURANTE

//MÉTODO POST

app.post('/api/guardar', (req, res) => {
    const { id_rest, nombre, ciudad, direccion, fecha_apertura } = req.body;
    const query = 'INSERT INTO Restaurante (id_rest, nombre, ciudad, direccion, fecha_apertura) VALUES ($1, $2, $3, $4, $5)';

    connection.query(query, [id_rest, nombre, ciudad, direccion, fecha_apertura], (error, result) => {
        if (error) {
            res.status(500).json({
                message: 'ERROR CREANDO EL RESTAURANTE',
                error
            });
        } else {
            res.status(201).json({ id_rest, nombre, ciudad, direccion, fecha_apertura });
        }
    });
});
```

Get

```
//MÉTODO GET

app.get('/api/obtener', (req, res) => {
    const query = 'SELECT * FROM Restaurante';

    connection.query(query, (error, result) => {
        if (error) {
            res.status(500).json({
                success: false,
                message: "Error al recuperar los datos",
                details: error.message
            });
        } else {
            res.status(200).json({
                success: true,
                message: "Datos de la tabla",
                details: result
            });
        }
    });
});
```

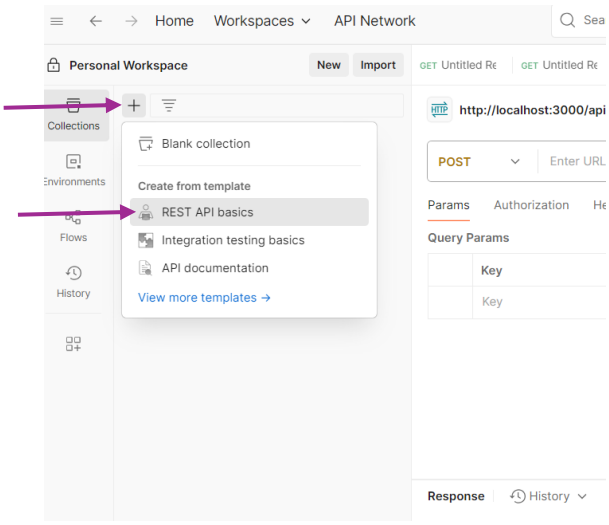
Put

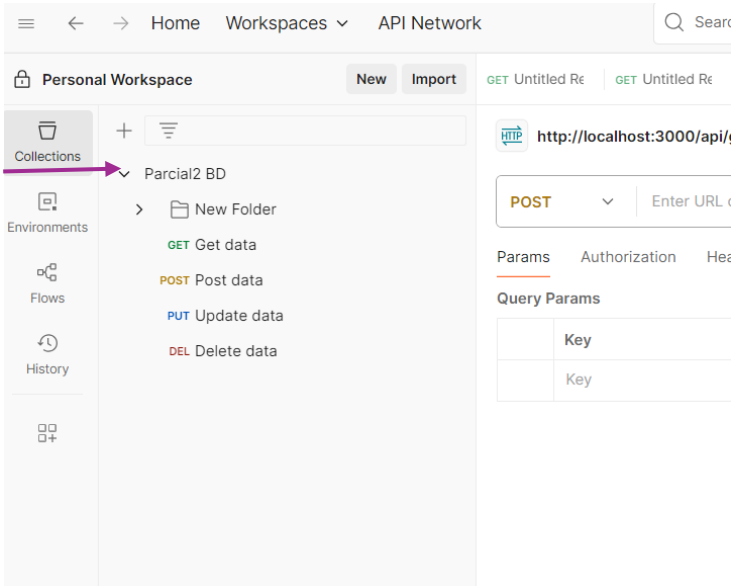
```
//MÉTODO PUT
app.put('/api/actualizar/:id_rest', (req, res) => {
    const { id_rest } = req.params;
    const { nombre, ciudad, direccion, fecha_apertura } = req.body;

    const query = 'UPDATE Restaurante SET nombre = $1, ciudad = $2, direccion = $3, fecha_apertura = $4 WHERE id_rest = $5';

    connection.query(query, [nombre, ciudad, direccion, fecha_apertura, id_rest], (error, result) => {
        if (error) {
            res.status(500).json({
                success: false,
                message: 'Error al actualizar el registro',
                details: error.message
            });
        } else if (result.rowCount === 0) {
            res.status(404).json({
                success: false,
                message: `No se encontró ningún registro con el id ${id_rest}`
            });
        } else {
            res.status(200).json({
                success: true,
                message: 'Registro actualizado correctamente',
                updated: {
                    id_rest,
                    nombre,
                    ciudad,
                    direccion,
                    fecha_apertura
                }
            });
        }
    });
```

Delete

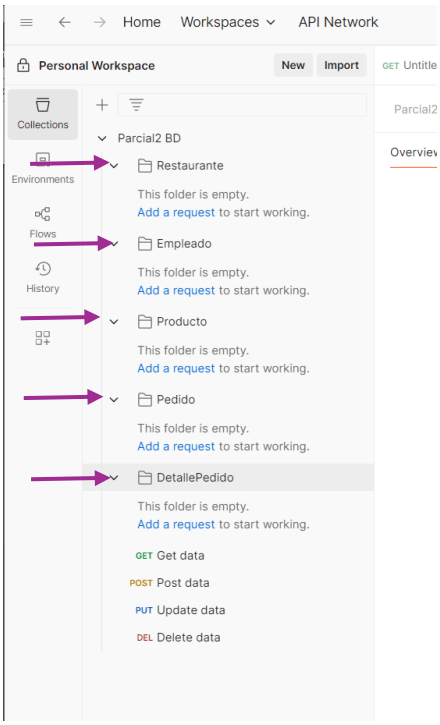
```
//MÉTODO DELETE

app.delete('/api/eliminar/:id_rest', (req, res) => {
    const { id_rest } = req.params;
    const query = 'DELETE FROM Restaurante WHERE id_rest = $1';

    connection.query(query, [id_rest], (error, result) => {

        if (error) {
            res.status(500).json({
                success: false,
                message: "Error al eliminar el registro",
                details: error.message
            });
        } else if (result.rowCount === 0) {
            res.status(404).json({
                success: false,
                message: `No existe el registro ${id_rest}`,

            });
        } else {
            res.status(200).json({
                success: true,
                message: "Dato eliminado de la tabla",
                details: result

            });
        }
    });
});
```

24. En la interfaz de Postman se crea una nueva colección, en este caso llamada ***“Parcial2 BD”.***



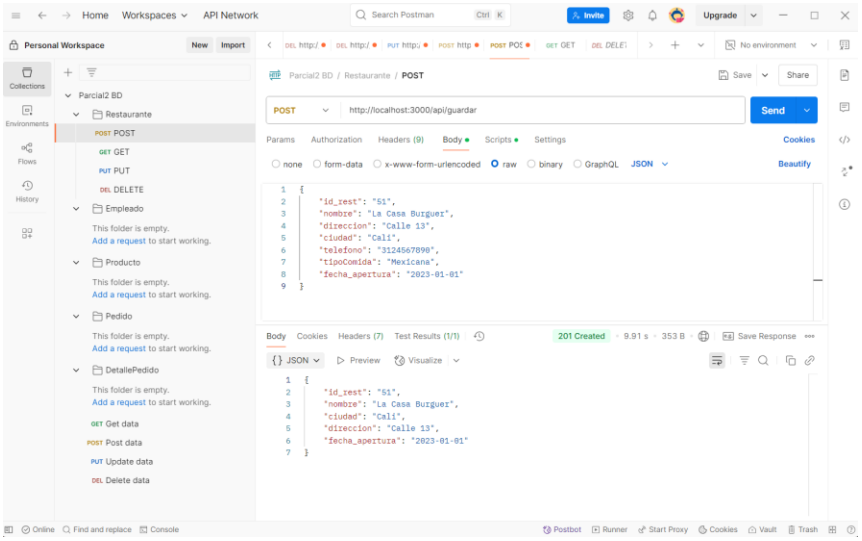


25. Dentro de la nueva colección ya creada, para mejor organización creamos una carpeta por cada CRUD a desarrollar.

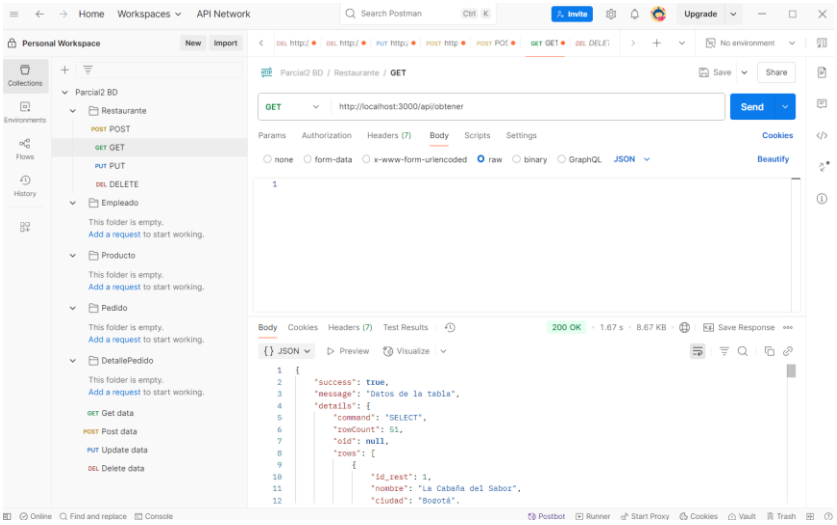


23. Probamos cada API en nuestro Postman de la tabla ***Restaurante***.

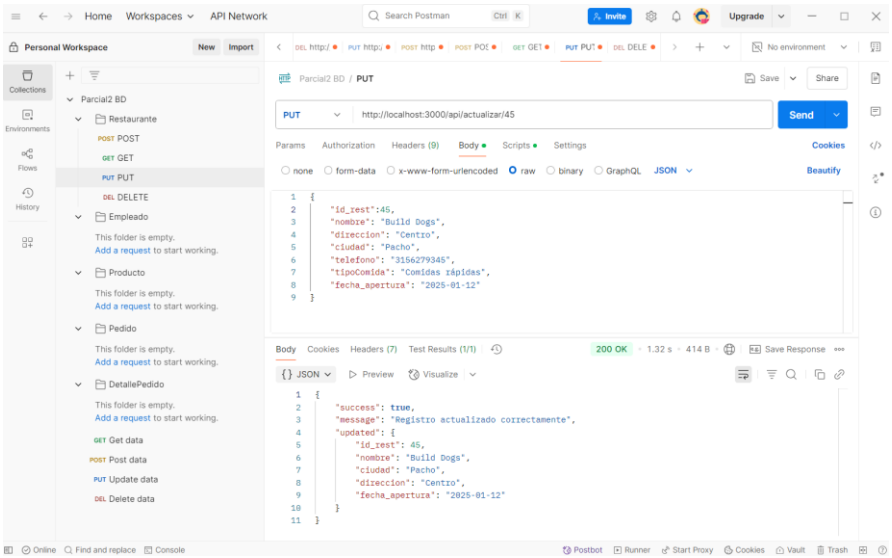
Post



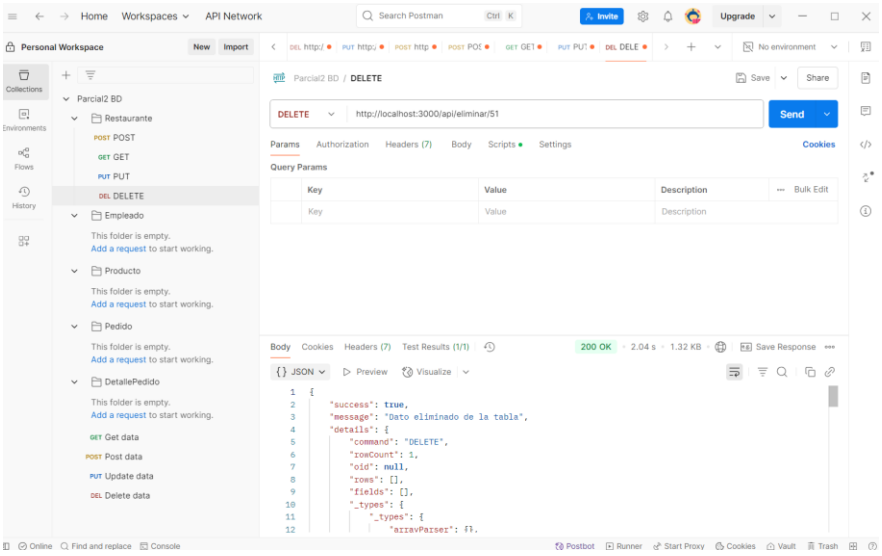
Get


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Personal Workspace
New
Import
Collections
Environments
Flows
History
Parcial2 BD
Restaurante
POST POST
GET GET
PUT PUT
DEL DELETE
Empleado
This folder is empty.
Add a request to start working.
Producto
Pedido
DetallePedido
GET Get data
POST Post data
PUT Update data
DEL Delete data
Parcial2 BD / Restaurante / GET
Save
Share
GET
http://localhost:3000/api/obtener
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
Test Results
200 OK
1.67 s
8.67 KB
Save Response
Preview
Visualize
{
"success": true,
"message": "Datos de la tabla",
"details": {
"command": "SELECT",
"rowCount": 51,
"oid": null,
"rows": [
{
"id_rest": 1,
"nombre": "La Cabaña del Sabor",
"ciudad": "Bogotá".
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

Put

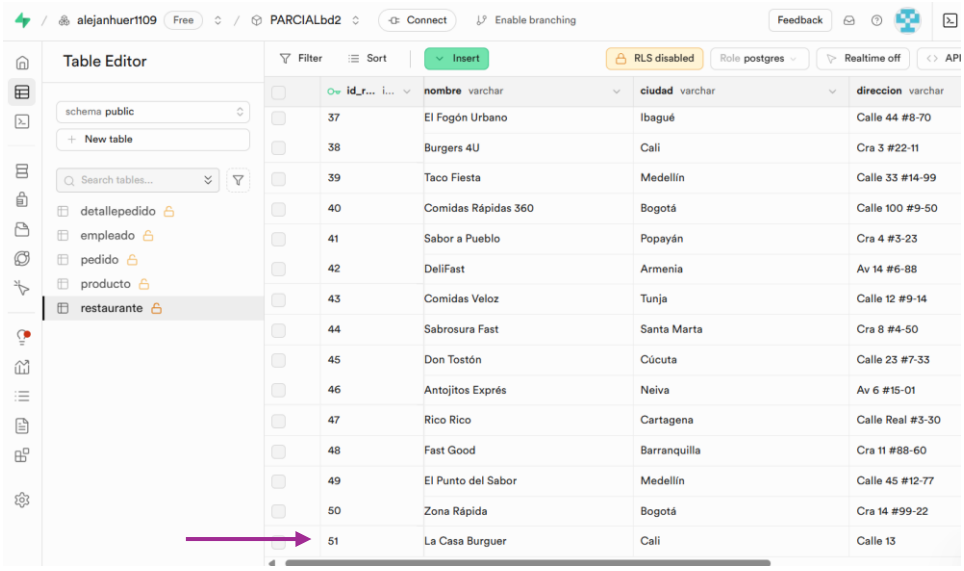

Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Personal Workspace
New
Import
Collections
Environments
Flows
History
Parcial2 BD
Restaurante
POST POST
GET GET
PUT PUT
DEL DELETE
Empleado
This folder is empty.
Add a request to start working.
Producto
Pedido
DetallePedido
GET Get data
POST Post data
PUT Update data
DEL Delete data
Parcial2 BD / PUT
Save
Share
PUT
http://localhost:3000/api/actualizar/45
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"id_rest":45,
"nombre": "Build Dogs",
"direccion": "Centro",
"ciudad": "Pacho",
"telefono": "3156279345",
"tipoComida": "Comidas rápidas",
"fecha_apertura": "2025-01-12"
}
Test Results (1/1)
200 OK
1.32 s
414 B
Save Response
Preview
Visualize
{
"success": true,
"message": "Registro actualizado correctamente",
"updated": {
"id_rest": 45,
"nombre": "Build Dogs",
"ciudad": "Pacho",
"direccion": "Centro",
"fecha_apertura": "2025-01-12"
}
}
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

Delete

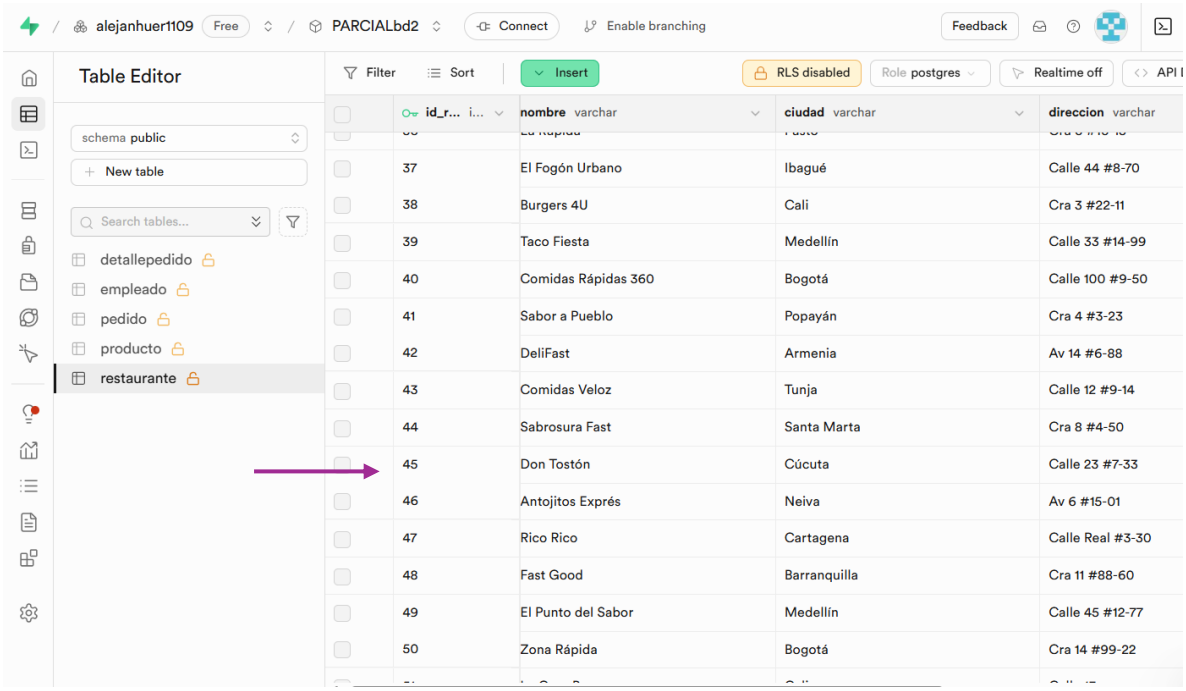


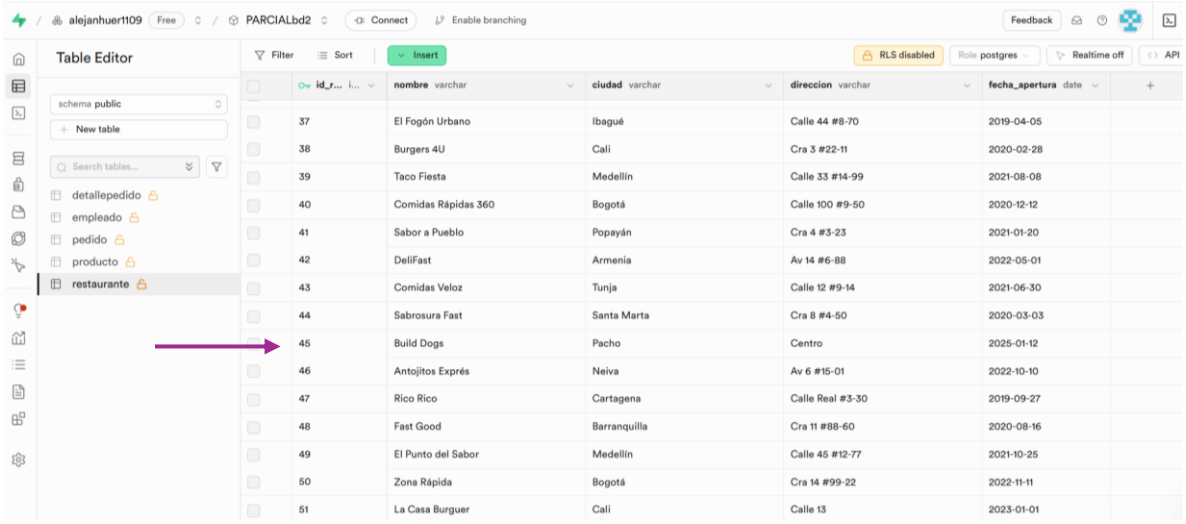
26. Verificamos la actualización de datos de la tabla ***Restaurante*** en nuestro Supabase.

Post

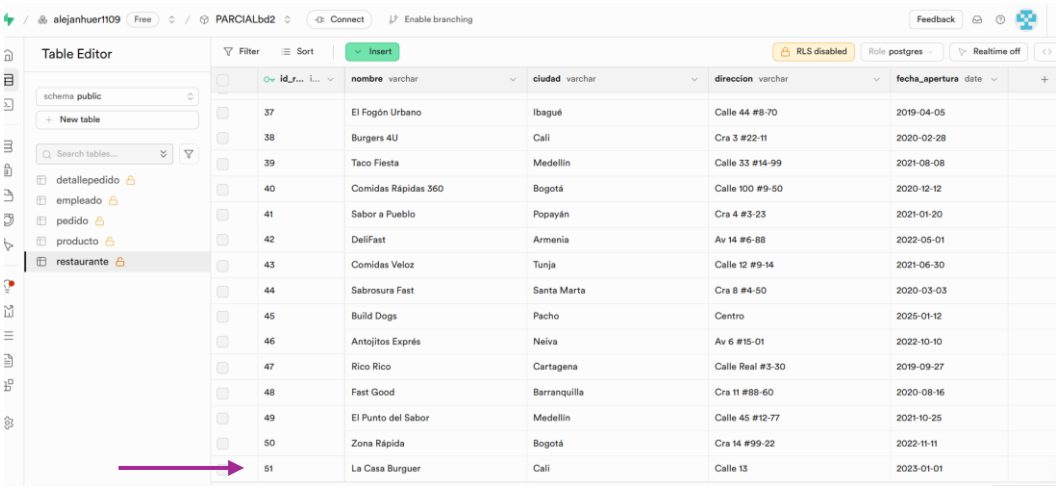


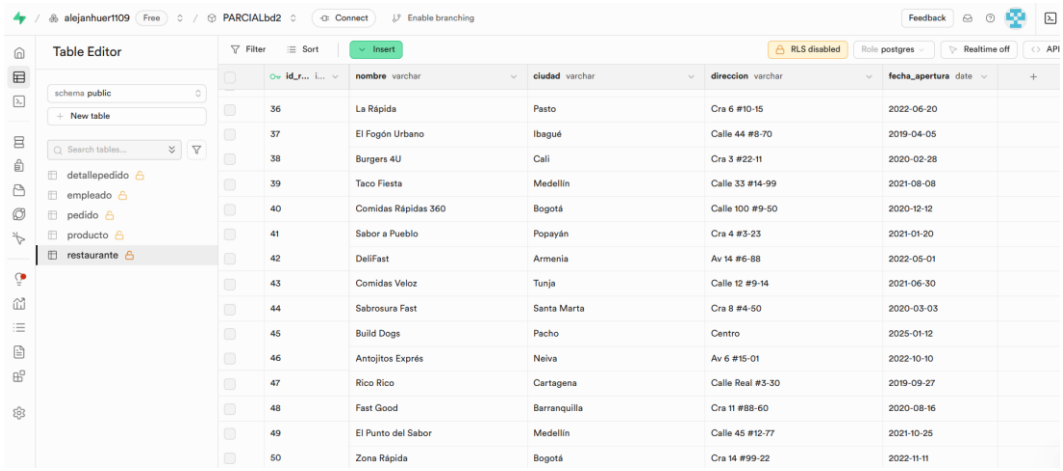
Put


alejanhuer1109
Free
PARCIALbd2
Connect
Enable branching
Feedback
Table Editor
Filter
Sort
Insert
RLS disabled
Role postgres
Realtime off
schema public
New table
Search tables...
detallepedido
empleado
pedido
producto
restaurante
id_r... i...
nombre varchar
ciudad varchar
direccion varchar
37 El Fogón Urbano Ibagué Calle 44 #8-70
38 Burgers 4U Cali Cra 3 #22-11
39 Taco Fiesta Medellín Calle 33 #14-99
40 Comidas Rápidas 360 Bogotá Calle 100 #9-50
41 Sabor a Pueblo Popayán Cra 4 #3-23
42 DeliFast Armenia Av 14 #6-88
43 Comidas Veloz Tunja Calle 12 #9-14
44 Sabrosura Fast Santa Marta Cra 8 #4-50
45 Don Tostón Cúcuta Calle 23 #7-33
46 Antojitos Exprés Neiva Av 6 #15-01
47 Rico Rico Cartagena Calle Real #3-30
48 Fast Good Barranquilla Cra 11 #88-60
49 El Punto del Sabor Medellín Calle 45 #12-77
50 Zona Rápida Bogotá Cra 14 #99-22


alejanhuer1109
Free
PARCIALbd2
Connect
Enable branching
Feedback
Table Editor
Filter
Sort
Insert
RLS disabled
Role postgres
Realtime off
schema public
New table
Search tables...
detallepedido
empleado
pedido
producto
restaurante
id_r... i...
nombre varchar
ciudad varchar
direccion varchar
fecha_apertura date
37 El Fogón Urbano Ibagué Calle 44 #8-70 2019-04-05
38 Burgers 4U Cali Cra 3 #22-11 2020-02-28
39 Taco Fiesta Medellín Calle 33 #14-99 2021-08-08
40 Comidas Rápidas 360 Bogotá Calle 100 #9-50 2020-12-12
41 Sabor a Pueblo Popayán Cra 4 #3-23 2021-01-20
42 DeliFast Armenia Av 14 #6-88 2022-05-01
43 Comidas Veloz Tunja Calle 12 #9-14 2021-06-30
44 Sabrosura Fast Santa Marta Cra 8 #4-50 2020-03-03
45 Build Dogs Pacho Centro 2025-01-12
46 Antojitos Exprés Neiva Av 6 #15-01 2022-10-10
47 Rico Rico Cartagena Calle Real #3-30 2019-09-27
48 Fast Good Barranquilla Cra 11 #88-60 2020-08-16
49 El Punto del Sabor Medellín Calle 45 #12-77 2021-10-25
50 Zona Rápida Bogotá Cra 14 #99-22 2022-11-11
51 La Casa Burguer Cali Calle 13 2023-01-01

Delete





27. Se crea el CRUD para la tabla ***Empleado*** en nuestro archivo index.js.

Post

```
149
150  //CRUD EMPLEADO
151
152  //MÉTODO POST
153
154  app.post('/api/guardar_empleado', (req, res) => {
155      const { id_empleado, nombre, rol, id_rest } = req.body;
156      const query = 'INSERT INTO Empleado (id_empleado, nombre, rol, id_rest) VALUES ($1, $2, $3, $4)';
157
158      connection.query(query, [id_empleado, nombre, rol, id_rest], (error, result) => {
159          if (error) {
160              res.status(500).json({
161                  message: 'ERROR CREANDO EL EMPLEADO',
162                  error
163              });
164          } else {
165              res.status(201).json({ id_empleado, nombre, rol, id_rest });
166          }
167      });
168  });
169
```

Get

```
171
172  //MÉTODO GET
173
174  app.get('/api/obtener_empleado', (req, res) => {
175      const query = 'SELECT * FROM Empleado';
176
177      connection.query(query, (error, result) => {
178          if (error) {
179              res.status(500).json({
180                  success: false,
181                  message: "Error al recuperar los datos",
182                  details: error.message
183              });
184          } else {
185              res.status(200).json({
186                  success: true,
187                  message: "Datos de la tabla",
188                  details: result
189              });
190          }
191      });
192  });
193
```

Put

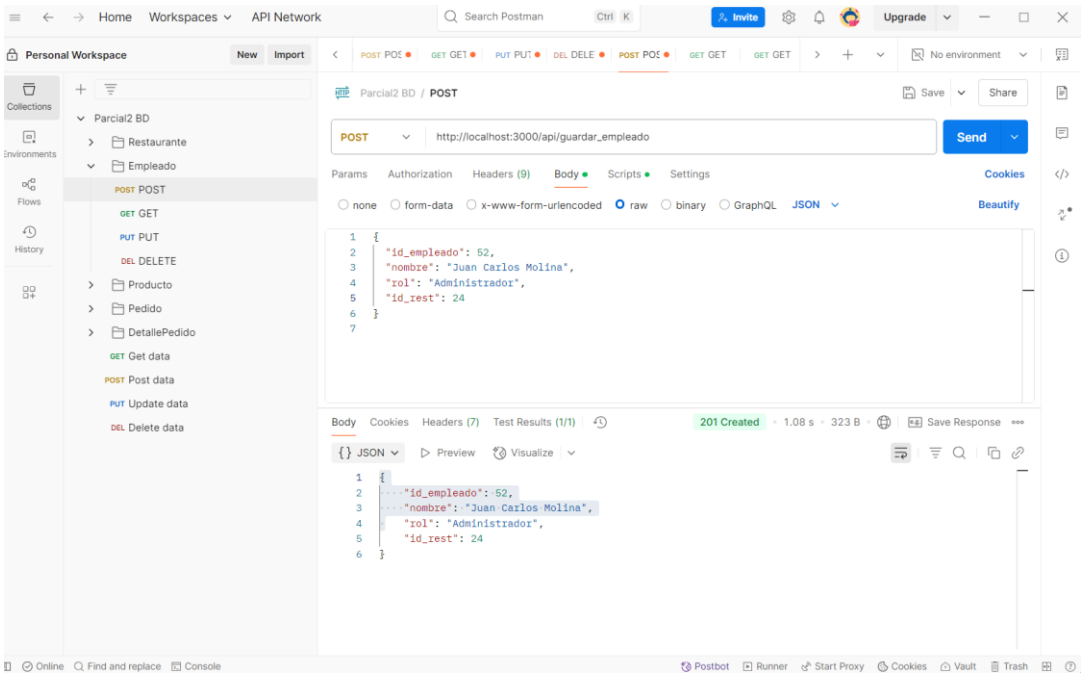
```
//METODO PUT

app.put('/api/actualizar_empleado/:id_empleado', (req, res) => {
    const { id_empleado } = req.params;
    const { nombre, rol, id_rest } = req.body;

    const query = 'UPDATE empleado SET nombre = $1, rol = $2, id_rest = $3 WHERE id_empleado = $4';

    connection.query(query, [nombre, rol, id_rest, id_empleado], (error, result) => {
        if (error) {
            res.status(500).json({
                success: false,
                message: 'Error al actualizar el registro',
                details: error.message
            });
        } else if (result.rowCount === 0) {
            res.status(404).json({
                success: false,
                message: `No se encontró ningún empleado con el id ${id_empleado}`
            });
        } else {
            res.status(200).json({
                success: true,
                message: 'Empleado actualizado correctamente',
                updated: {
                    id_empleado,
                    nombre,
                    rol,
                    id_rest
                }
            });
        }
    });
```

Delete

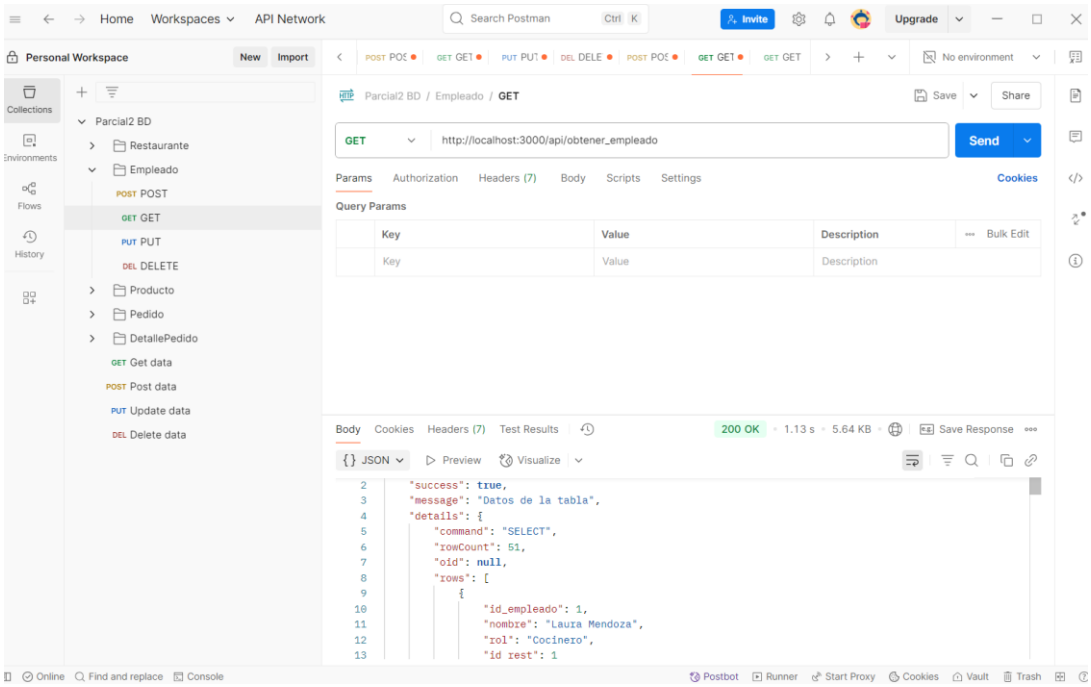
```
//MÉTODO DELETE

app.delete('/api/eliminar_empleado/:id_empleado', (req, res) => {
    const { id_empleado } = req.params;
    const query = 'DELETE FROM Empleado WHERE id_empleado = $1';

    connection.query(query, [id_empleado], (error, result) => {

        if (error) {
            res.status(500).json({
                success: false,
                message: "Error al eliminar el registro",
                details: error.message
            });
        } else if (result.rowCount === 0) {
            res.status(404).json({
                success: false,
                message: `No existe el registro ${id_empleado}`,

            });
        } else {
            res.status(200).json({
                success: true,
                message: "Dato eliminado de la tabla",
                details: result

            });
        }
    });
});
```

28. Probamos cada API en nuestro Postman de la tabla ***Empleado***.

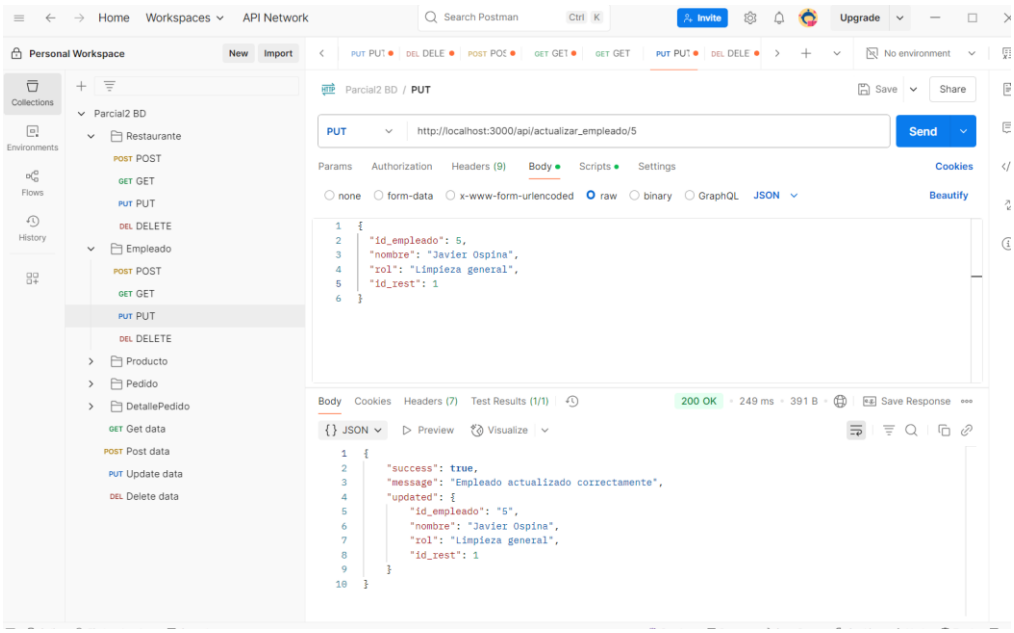
Post


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Personal Workspace
New
Import
No environment
Collections
Environments
Flows
History
Parcial2 BD
Restaurante
Empleado
POST POST
GET GET
PUT PUT
DEL DELETE
Producto
Pedido
DetallePedido
GET Get data
POST Post data
PUT Update data
DEL Delete data
Parcial2 BD / POST
Save
Share
POST
http://localhost:3000/api/guardar_empleado
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"id_empleado": 52,
"nombre": "Juan Carlos Molina",
"rol": "Administrador",
"id_rest": 24
}
Body
Cookies
Headers (7)
Test Results (1/1)
201 Created
1.08 s
323 B
Save Response
JSON
Preview
Visualize
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

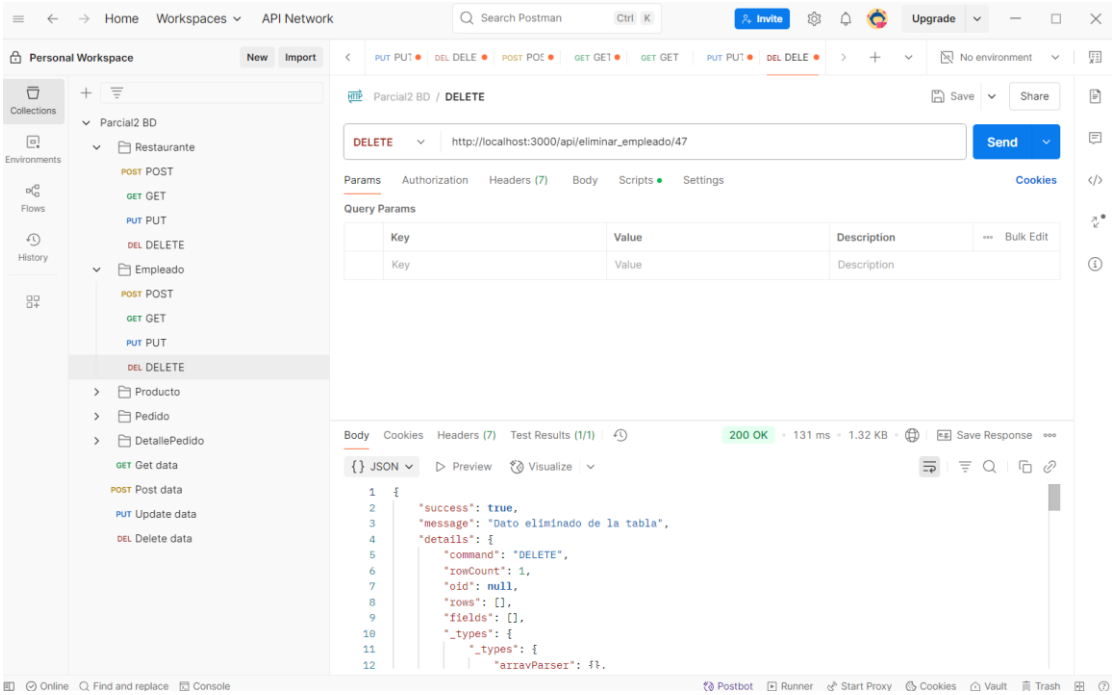
Get


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Personal Workspace
New
Import
No environment
Collections
Environments
Flows
History
Parcial2 BD
Restaurante
Empleado
POST POST
GET GET
PUT PUT
DEL DELETE
Producto
Pedido
DetallePedido
GET Get data
POST Post data
PUT Update data
DEL Delete data
Parcial2 BD / Empleado / GET
Save
Share
GET
http://localhost:3000/api/obtener_empleado
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (7)
Test Results
200 OK
1.13 s
5.64 KB
Save Response
JSON
Preview
Visualize
"success": true,
"message": "Datos de la tabla",
"details": {
"command": "SELECT",
"rowCount": 51,
"oid": null,
"rows": [
{
"id_empleado": 1,
"nombre": "Laura Mendoza",
"rol": "Cocinero",
"id_rest": 1
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

Put

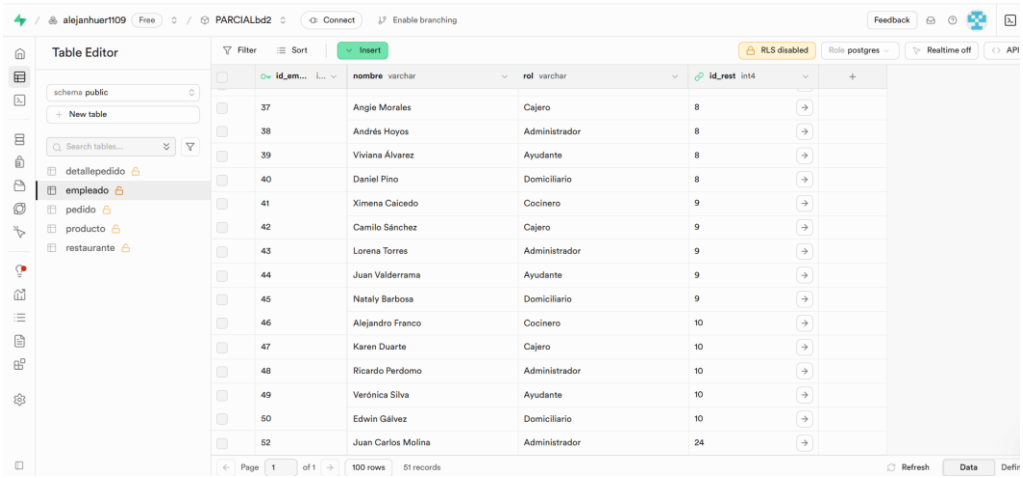


Delete

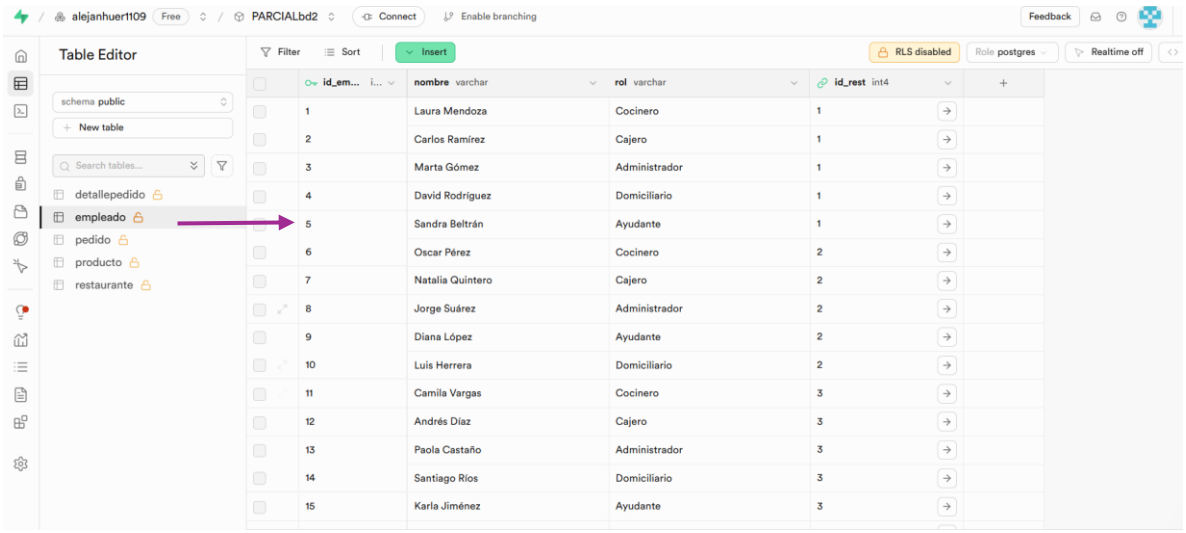


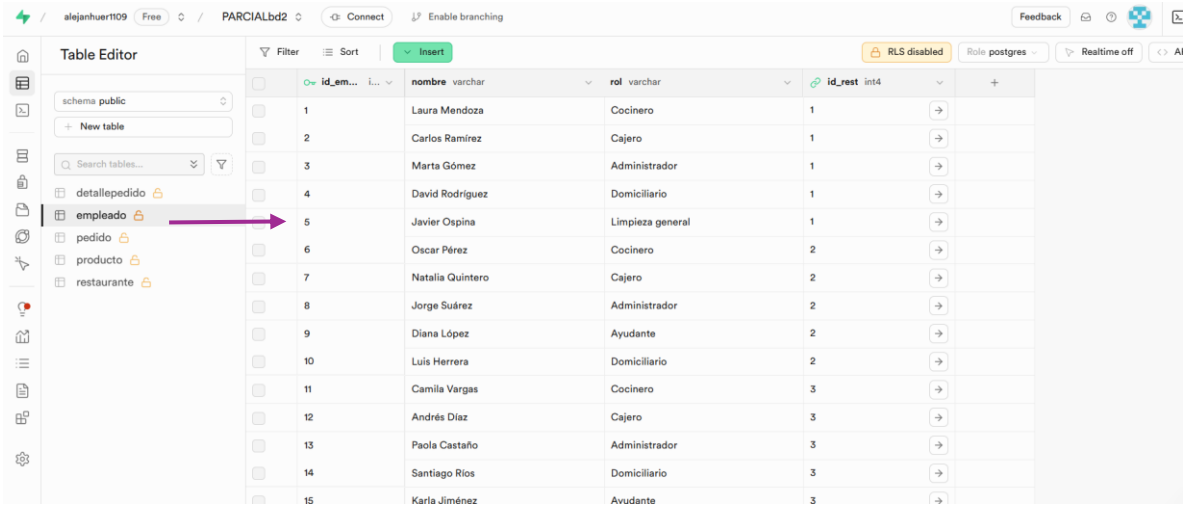
29. Verificamos la actualización de datos de la tabla ***Empleado*** en nuestro Supabase.

Post

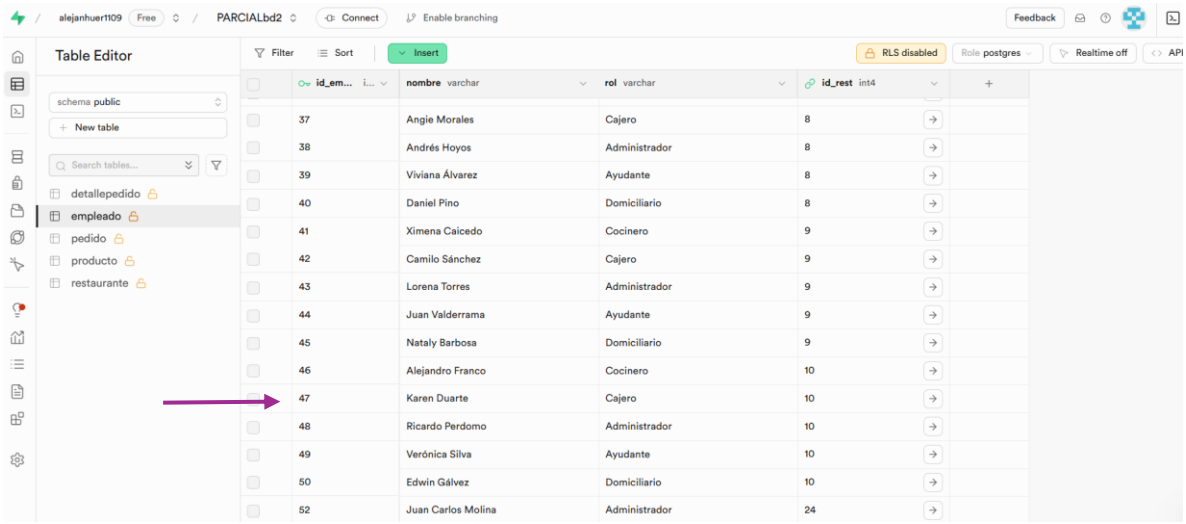

alejanhuer1109
Free
PARCIALbd2
Connect
Enable branching
Feedback
Table Editor
Filter
Sort
Insert
RLS disabled
Role postgres
Realtime off
schema public
New table
Search tables...
detallepedido
empleado
pedido
producto
restaurante
id_em... int4
nombre varchar
rol varchar
id_rest int4
37 Angie Morales Cajero 8
38 Andrés Hoyos Administrador 8
39 Viviana Álvarez Ayudante 8
40 Daniel Pino Domiciliario 8
41 Ximena Caicedo Cocinero 9
42 Camilo Sánchez Cajero 9
43 Lorena Torres Administrador 9
44 Juan Valderrama Ayudante 9
45 Nataly Barbosa Domiciliario 9
46 Alejandro Franco Cocinero 10
47 Karen Duarte Cajero 10
48 Ricardo Perdomo Administrador 10
49 Verónica Silva Ayudante 10
50 Edwin Gálvez Domiciliario 10
52 Juan Carlos Molina Administrador 24
Page 1 of 1
100 rows
51 records
Refresh
Data

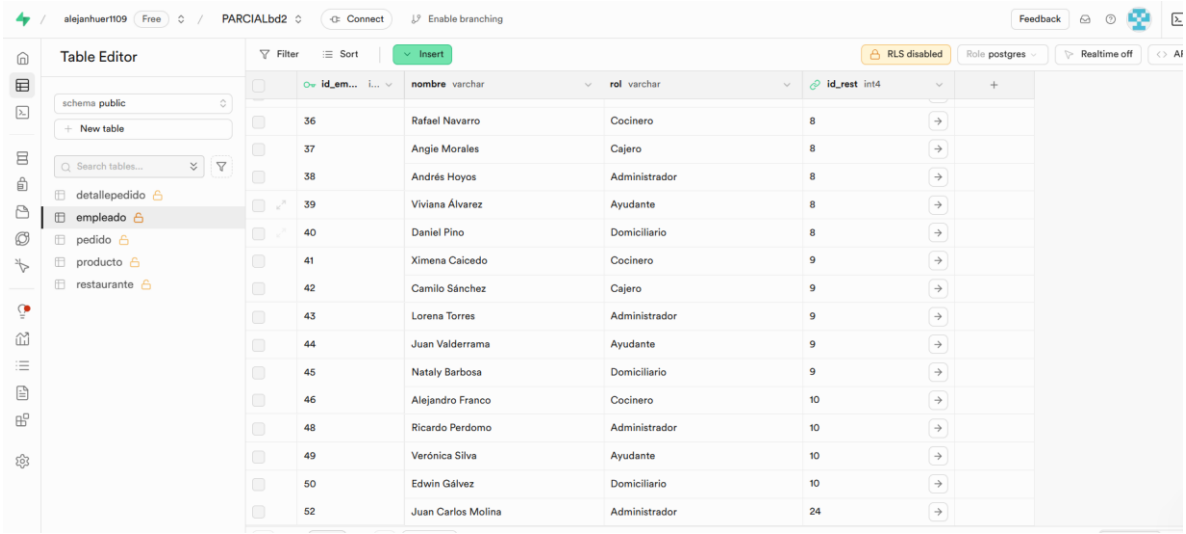
Put


alejanhuer1109
Free
PARCIALbd2
Connect
Enable branching
Feedback
Table Editor
Filter
Sort
Insert
RLS disabled
Role postgres
Realtime off
schema public
New table
Search tables...
detallepedido
empleado
pedido
producto
restaurante
id_em... int4
nombre varchar
rol varchar
id_rest int4
1 Laura Mendoza Cocinero 1
2 Carlos Ramírez Cajero 1
3 Marta Gómez Administrador 1
4 David Rodríguez Domiciliario 1
5 Sandra Beltrán Ayudante 1
6 Oscar Pérez Cocinero 2
7 Natalia Quintero Cajero 2
8 Jorge Suárez Administrador 2
9 Diana López Ayudante 2
10 Luis Herrera Domiciliario 2
11 Camila Vargas Cocinero 3
12 Andrés Díaz Cajero 3
13 Paola Castaño Administrador 3
14 Santiago Ríos Domiciliario 3
15 Karla Jiménez Ayudante 3


alejanhuer1109
Free
PARCIALbd2
Connect
Enable branching
Feedback
Table Editor
Filter
Sort
Insert
RLS disabled
Role postgres
Realtime off
schema public
New table
Search tables...
detallepedido
empleado
pedido
producto
restaurante
id_em... int4
nombre varchar
rol varchar
id_rest int4
1 Laura Mendoza Cocinero 1
2 Carlos Ramírez Cajero 1
3 Marta Gómez Administrador 1
4 David Rodríguez Domiciliario 1
5 Javier Ospina Limpieza general 1
6 Oscar Pérez Cocinero 2
7 Natalia Quintero Cajero 2
8 Jorge Suárez Administrador 2
9 Diana López Ayudante 2
10 Luis Herrera Domiciliario 2
11 Camila Vargas Cocinero 3
12 Andrés Díaz Cajero 3
13 Paola Castaño Administrador 3
14 Santiago Ríos Domiciliario 3
15 Karla Jiménez Ayudante 3

Delete





30. Se crea el CRUD para la tabla ***Producto*** en nuestro archivo index.js.

Post

```
262  //CRUD PRODUCTO
263
264  //MÉTODO POST
265
266  app.post('/api/guardar_producto', (req, res) => {
267      const { id_prod, nombre, precio } = req.body;
268      const query = 'INSERT INTO Producto (id_prod, nombre, precio) VALUES ($1, $2, $3)';
269
270      connection.query(query, [id_prod, nombre, precio], (error, result) => {
271          if (error) {
272              res.status(500).json({
273                  message: 'ERROR CREANDO EL PRODUCTO',
274                  error
275              });
276          } else {
277              res.status(201).json({ id_prod, nombre, precio });
278          }
279      });
280  });
281
```

Get

```
283
284  //MÉTODO GET
285
286  app.get('/api/obtener_producto', (req, res) => {
287      const query = 'SELECT * FROM Producto';
288
289      connection.query(query, (error, result) => {
290          if (error) {
291              res.status(500).json({
292                  success: false,
293                  message: "Error al recuperar los datos",
294                  details: error.message
295              });
296          } else {
297              res.status(200).json({
298                  success: true,
299                  message: "Datos de la tabla",
300                  details: result
301              });
302          }
303      });
304  });
305
```

Put

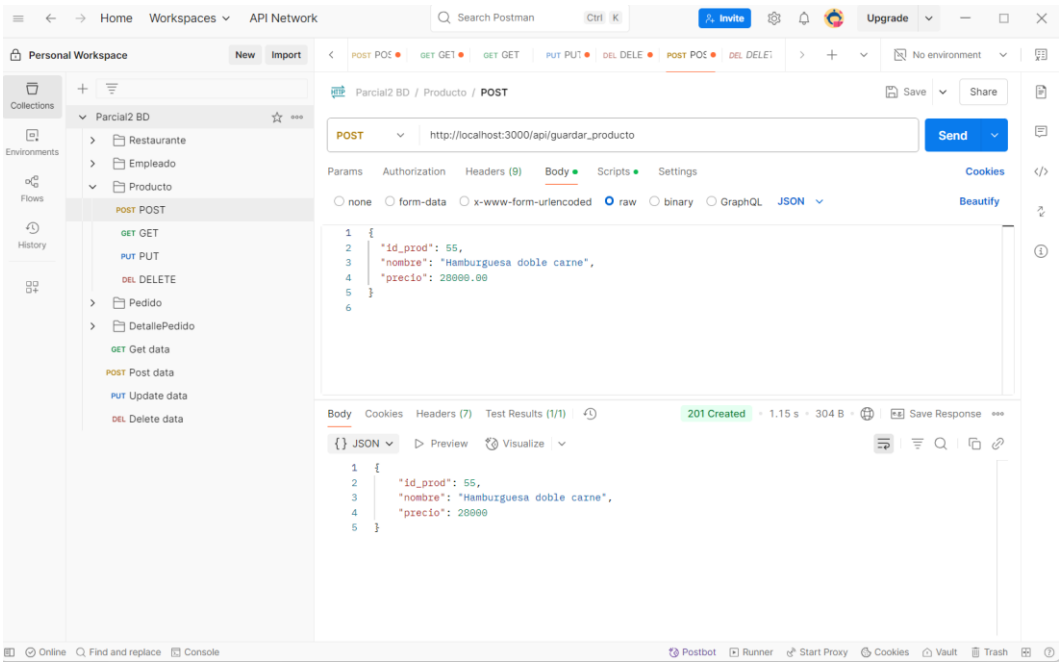
```
306  //MÉTODO PUT
307
308  app.put('/api/actualizar_producto/:id_prod', (req, res) => {
309      const { id_prod } = req.params;
310      const { nombre, precio } = req.body;
311
312      const query = 'UPDATE Producto SET nombre = $1, precio = $2 WHERE id_prod = $3';
313
314      connection.query(query, [nombre, precio, id_prod], (error, result) => {
315          if (error) {
316              res.status(500).json({
317                  success: false,
318                  message: 'Error al actualizar el registro',
319                  details: error.message
320              });
321          } else if (result.rowCount === 0) {
322              res.status(404).json({
323                  success: false,
324                  message: `No se encontró ningún producto con el id ${id_prod}`
325              });
326          } else {
327              res.status(200).json({
328                  success: true,
329                  message: 'Producto actualizado correctamente',
330                  updated: {
331                      id_prod,
332                      nombre,
333                      precio
334                  }
335              });
336          }
337      });
338  });
```

Delete

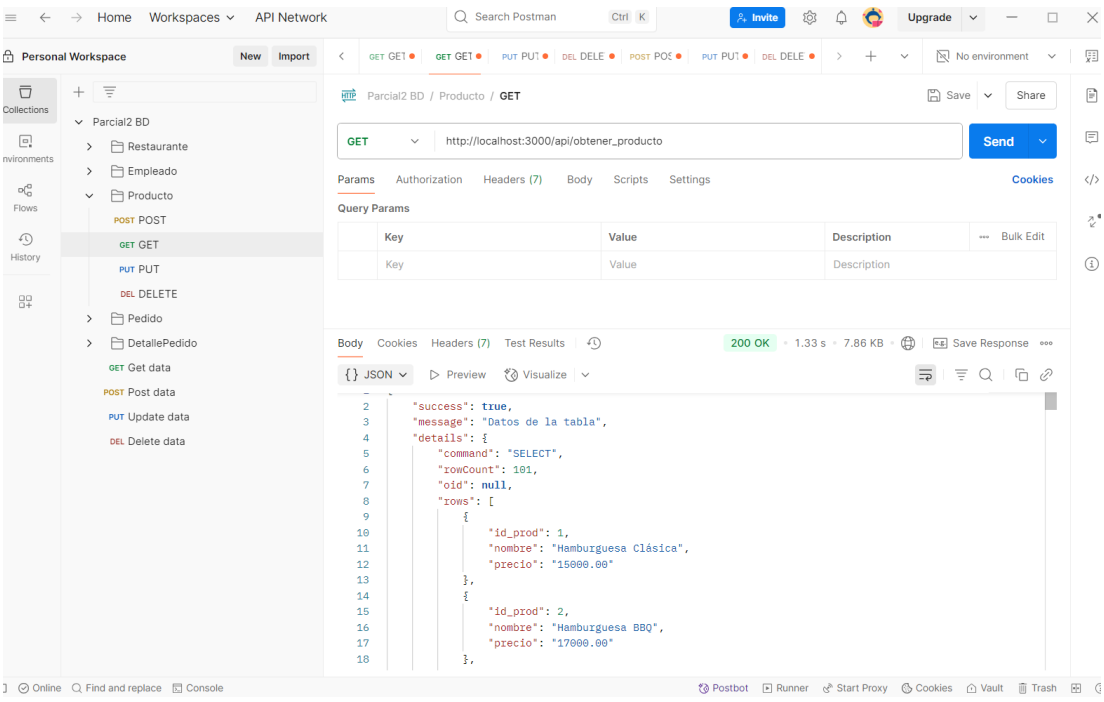
```
342  //MÉTODO DELETE
343
344  app.delete('/api/eliminar_producto/:id_prod', (req, res) => {
345      const { id_prod} = req.params;
346      const query = 'DELETE FROM Producto WHERE id_prod = $1';
347
348      connection.query(query, [id_prod], (error, result) => {
349
350          if (error) {
351              res.status(500).json({
352                  success: false,
353                  message: "Error al eliminar el producto",
354                  details: error.message
355              });
356          } else if (result.rowCount === 0) {
357              res.status(404).json({
358                  success: false,
359                  message: `No existe el producto ${id_prod}`,
360
361              });
362          } else {
363              res.status(200).json({
364                  success: true,
365                  message: "Producto eliminado de la tabla",
366                  details: result
367
368              });
369          }
370      });
371  });
```

31. Probamos cada API en nuestro Postman de la tabla ***Producto***.

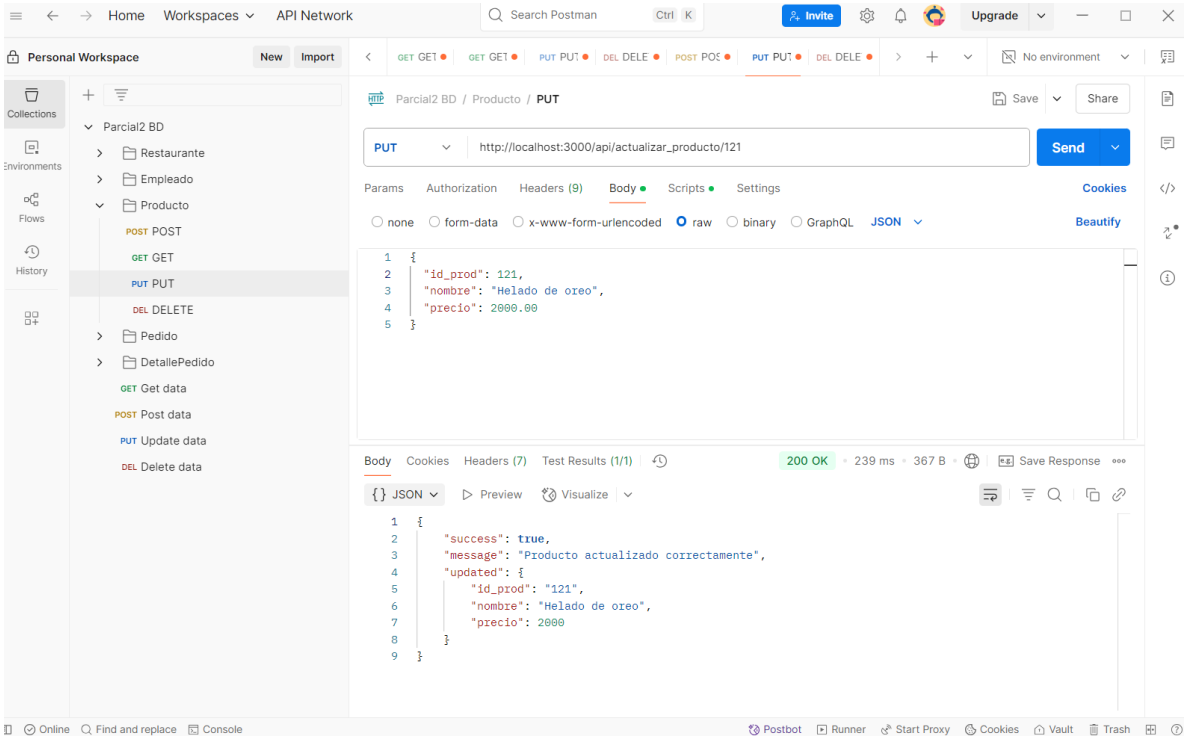
Post



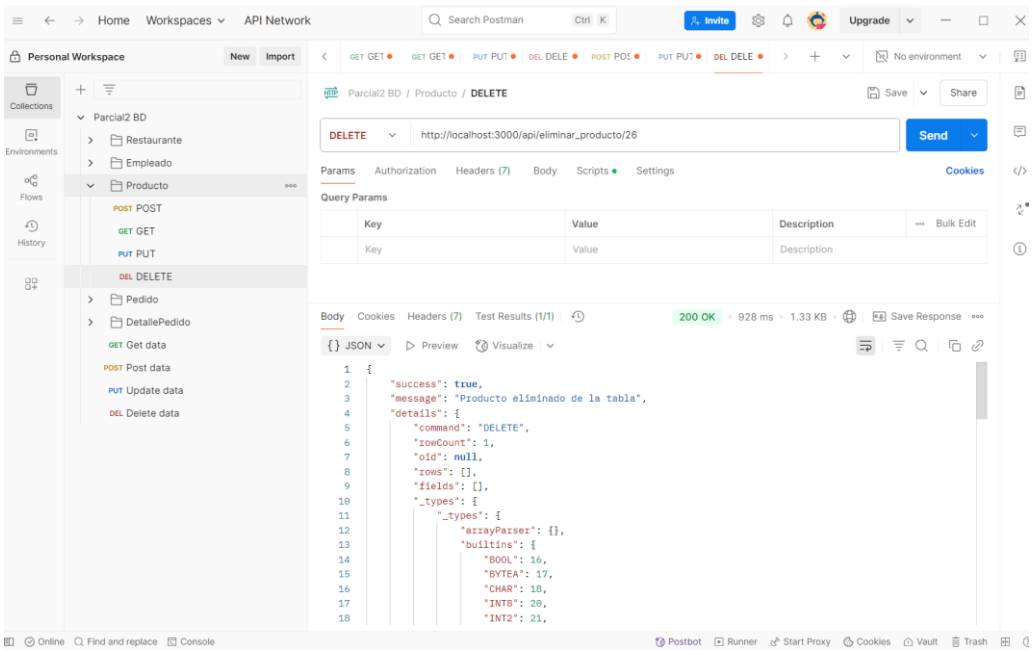
Get



Put

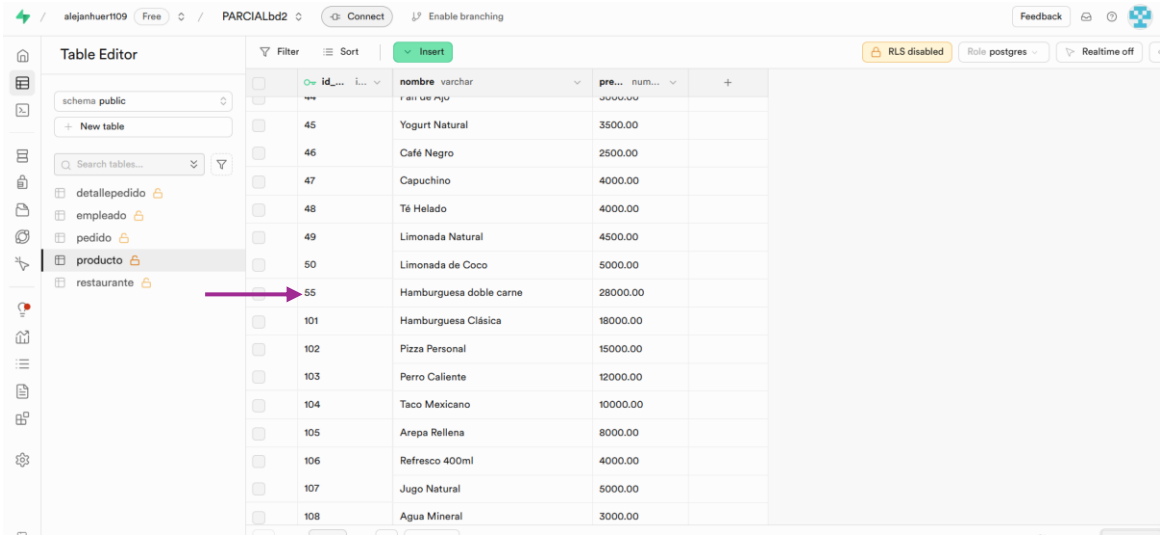

Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Personal Workspace
New
Import
No environment
Collections
Environments
Flows
History
Parcial2 BD
Restaurante
Empleado
Producto
POST POST
GET GET
PUT PUT
DEL DELETE
Pedido
DetallePedido
GET Get data
POST Post data
PUT Update data
DEL Delete data
Parcial2 BD / Producto / PUT
Save
Share
PUT
http://localhost:3000/api/actualizar_producto/121
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"id_prod": 121,
"nombre": "Helado de oreo",
"precio": 2000.00
}
Body
Cookies
Headers (7)
Test Results (1/1)
200 OK
239 ms
367 B
Save Response
JSON
Preview
Visualize
{
"success": true,
"message": "Producto actualizado correctamente",
"updated": {
"id_prod": "121",
"nombre": "Helado de oreo",
"precio": 2000
}
}
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

Delete

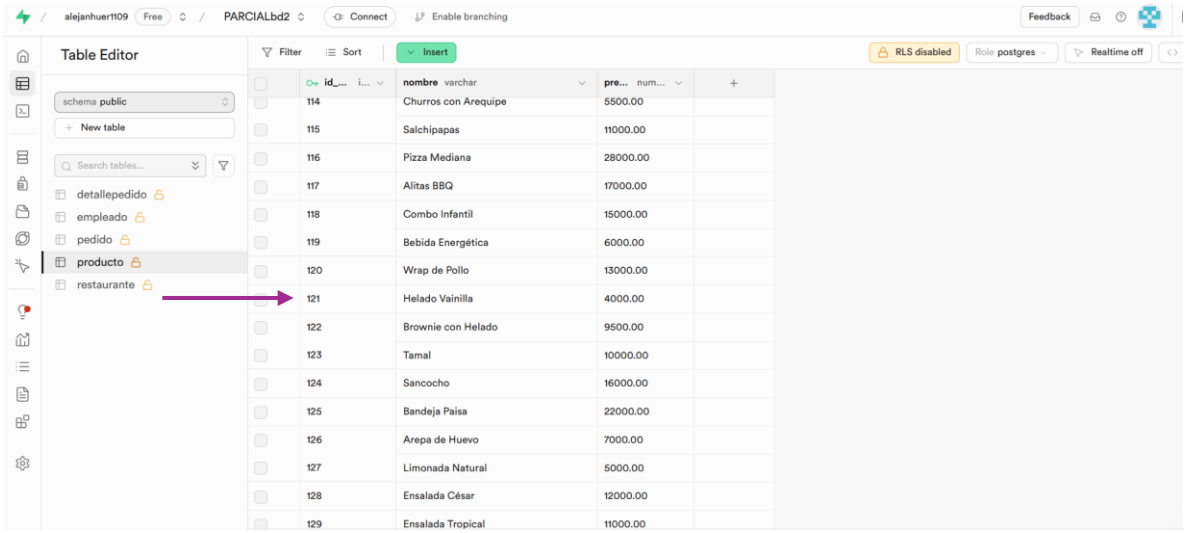

Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Personal Workspace
New
Import
No environment
Collections
Environments
Flows
History
Parcial2 BD
Restaurante
Empleado
Producto
POST POST
GET GET
PUT PUT
DEL DELETE
Pedido
DetallePedido
GET Get data
POST Post data
PUT Update data
DEL Delete data
Parcial2 BD / Producto / DELETE
Save
Share
DELETE
http://localhost:3000/api/eliminar_producto/26
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (7)
Test Results (1/1)
200 OK
928 ms
1.33 KB
Save Response
JSON
Preview
Visualize
{
"success": true,
"message": "Producto eliminado de la tabla",
"details": {
"command": "DELETE",
"rowCount": 1,
"oid": null,
"rows": [],
"fields": [],
"_types": {
"_types": {
"arrayParser": {},
"builtins": {
"BOOL": 16,
"BYTEA": 17,
"CHAR": 18,
"INT8": 20,
"INT2": 21,
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

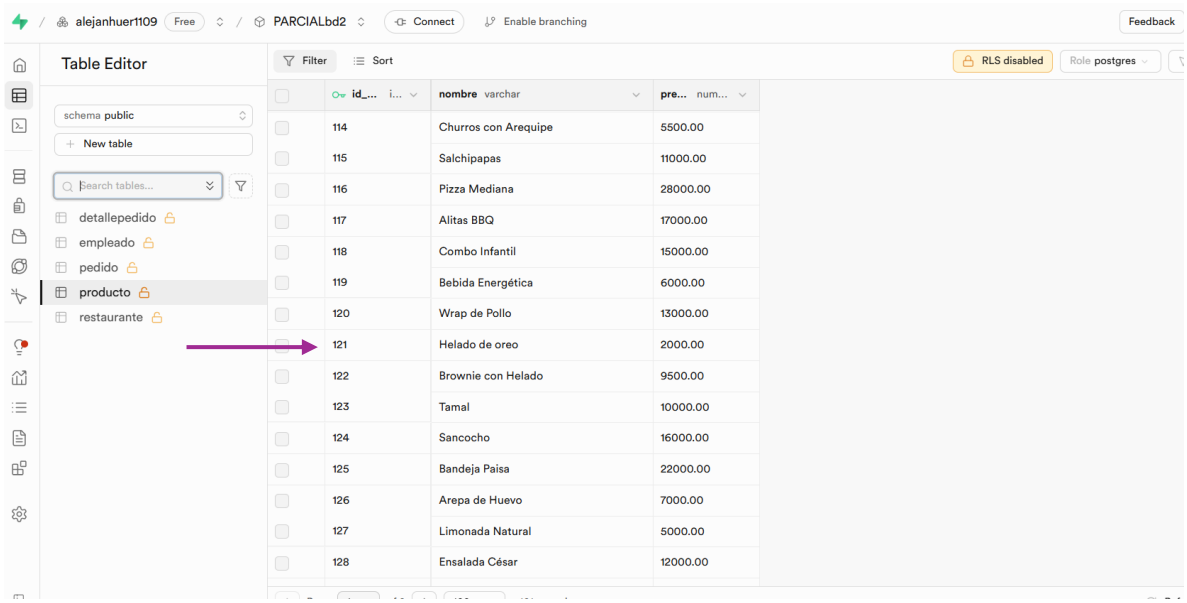
32. Verificamos la actualización de datos de la tabla ***Producto*** en nuestro Supabase.

Post

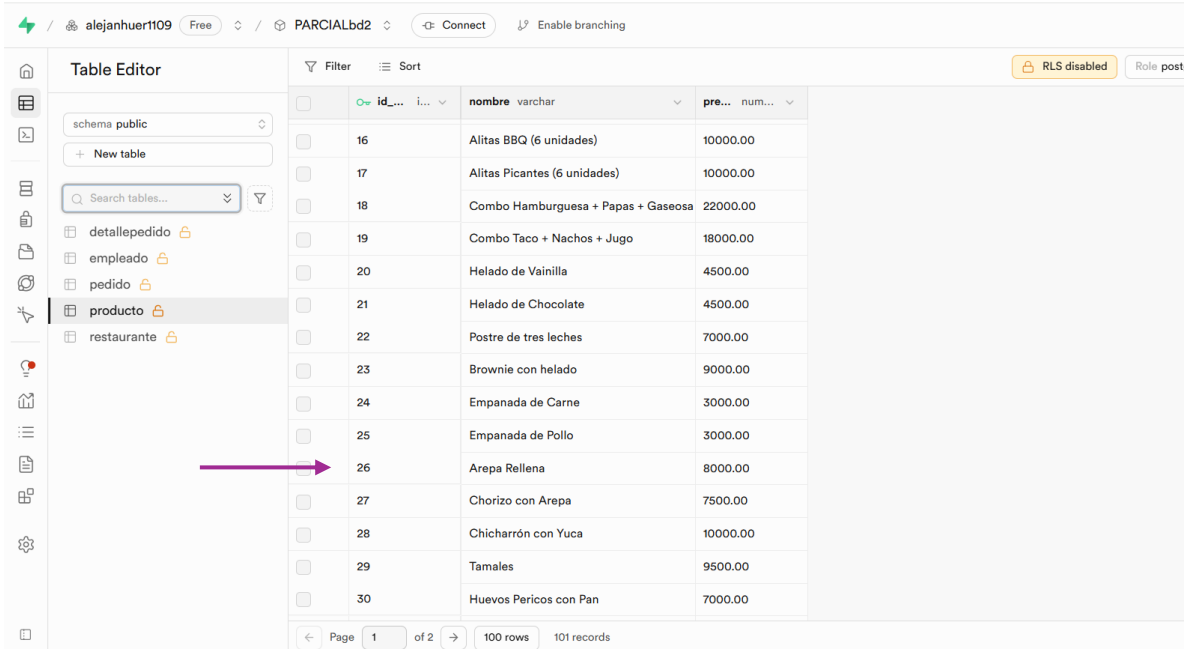


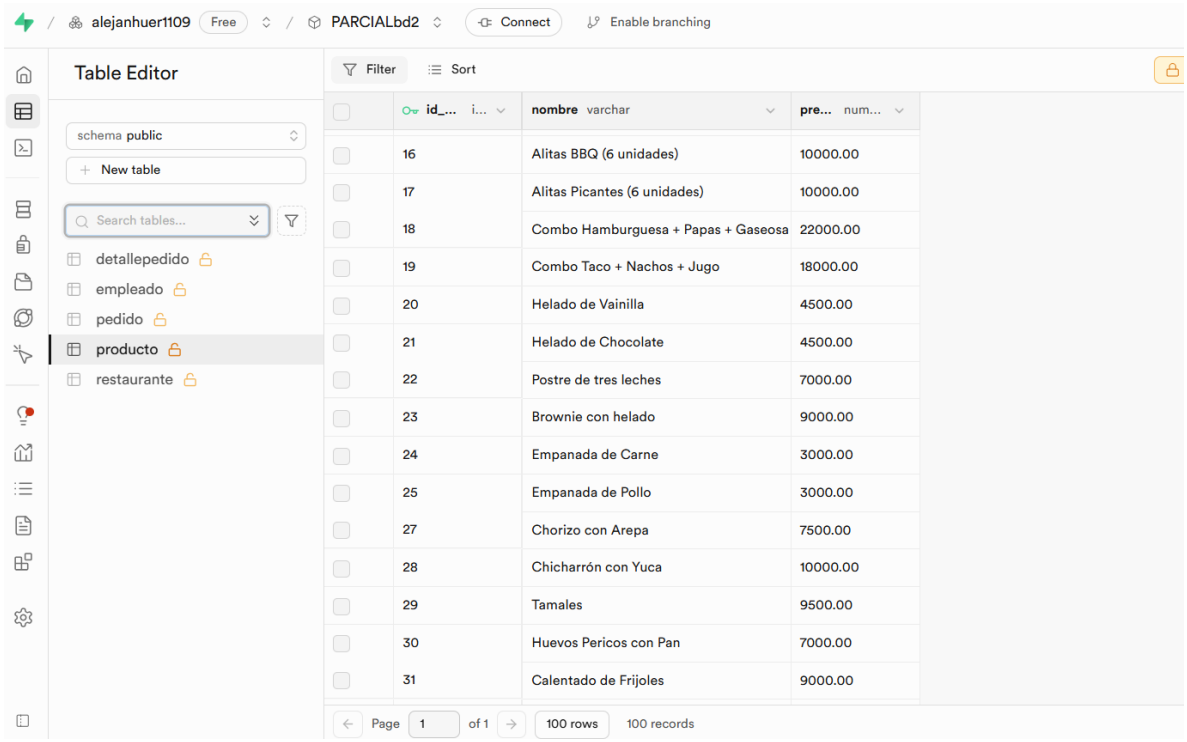
Put




alejanhuer1109
Free
PARCIALbd2
Connect
Enable branching
Feedback
Table Editor
Filter
Sort
RLS disabled
Role postgres
schema public
New table
Search tables...
detallepedido
empleado
pedido
producto
restaurante
id_... i...
nombre varchar
pre... num...
114 Churros con Arequipe 5500.00
115 Salchipapas 11000.00
116 Pizza Mediana 28000.00
117 Alitas BBQ 17000.00
118 Combo Infantil 15000.00
119 Bebida Energética 6000.00
120 Wrap de Pollo 13000.00
121 Helado de oreo 2000.00
122 Brownie con Helado 9500.00
123 Tamal 10000.00
124 Sancocho 16000.00
125 Bandeja Paisa 22000.00
126 Arepa de Huevo 7000.00
127 Limonada Natural 5000.00
128 Ensalada César 12000.00

Delete


alejanhuer1109
Free
PARCIALbd2
Connect
Enable branching
Table Editor
Filter
Sort
RLS disabled
schema public
New table
Search tables...
detallepedido
empleado
pedido
producto
restaurante
id_... i...
nombre varchar
pre... num...
16 Alitas BBQ (6 unidades) 10000.00
17 Alitas Picantes (6 unidades) 10000.00
18 Combo Hamburguesa + Papas + Gaseosa 22000.00
19 Combo Taco + Nachos + Jugo 18000.00
20 Helado de Vainilla 4500.00
21 Helado de Chocolate 4500.00
22 Postre de tres leches 7000.00
23 Brownie con helado 9000.00
24 Empanada de Carne 3000.00
25 Empanada de Pollo 3000.00
26 Arepa Rellena 8000.00
27 Chorizo con Arepa 7500.00
28 Chicharrón con Yuca 10000.00
29 Tamales 9500.00
30 Huevos Pericos con Pan 7000.00
Page 1 of 2
100 rows
101 records



33. Se crea el CRUD para la tabla ***Pedido*** en nuestro archivo index.js.

Post

```
//CRUD PEDIDO

//MÉTODO POST

app.post('/api/guardar_pedido', (req, res) => {
    const { id_pedido, fecha, id_rest, total } = req.body;
    const query = 'INSERT INTO Pedido (id_pedido, fecha, id_rest, total) VALUES ($1, $2, $3, $4)';

    connection.query(query, [id_pedido, fecha, id_rest, total], (error, result) => {
        if (error) {
            res.status(500).json({
                message: 'ERROR CREANDO EL PEDIDO',
                error
            });
        } else {
            res.status(201).json({ id_pedido, fecha, id_rest, total });
        }
    });
});

```

Get

```
395  //MÉTODO GET
396
397  app.get('/api/obtener_pedido', (req, res) => {
398      const query = 'SELECT * FROM Pedido';
399
400      connection.query(query, (error, result) => {
401          if (error) {
402              res.status(500).json({
403                  success: false,
404                  message: "Error al recuperar los datos de el pedido",
405                  details: error.message
406              });
407          } else {
408              res.status(200).json({
409                  success: true,
410                  message: "Datos de el pedido",
411                  details: result
412              });
413          }
414      });
415  });
```

Put

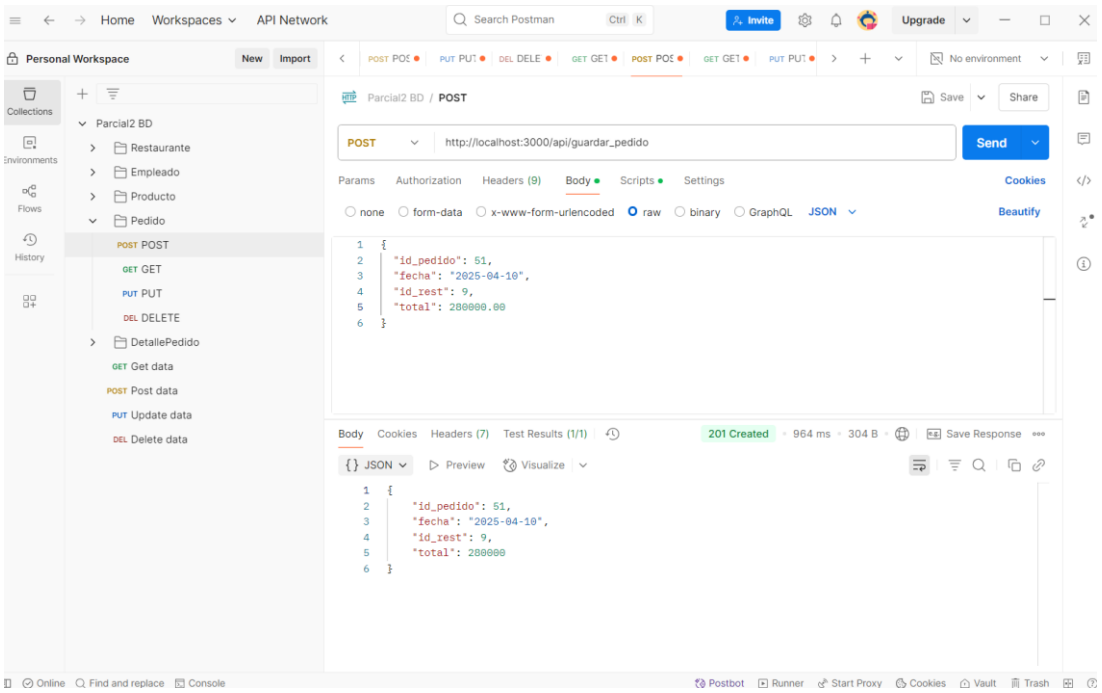
```
417  //METODO PUT
418  app.put('/api/actualizar_pedido/:id_pedido', (req, res) => {
419      const { id_pedido } = req.params;
420      const { fecha, id_rest, total } = req.body;
421
422      const query = 'UPDATE Pedido SET fecha = $1, id_rest = $2, total = $3 WHERE id_pedido = $4';
423
424      connection.query(query, [fecha, id_rest, total, id_pedido], (error, result) => {
425          if (error) {
426              res.status(500).json({
427                  success: false,
428                  message: 'Error al actualizar el pedido',
429                  details: error.message
430              });
431          } else if (result.rowCount === 0) {
432              res.status(404).json({
433                  success: false,
434                  message: `No se encontró ningún pedido con el id ${id_pedido}`
435              });
436          } else {
437              res.status(200).json({
438                  success: true,
439                  message: 'Pedido actualizado correctamente',
440                  updated: {
441                      id_pedido,
442                      fecha,
443                      id_rest,
444                      total
445                  }
446              });
447          }
448      });
449  });
```

Delete

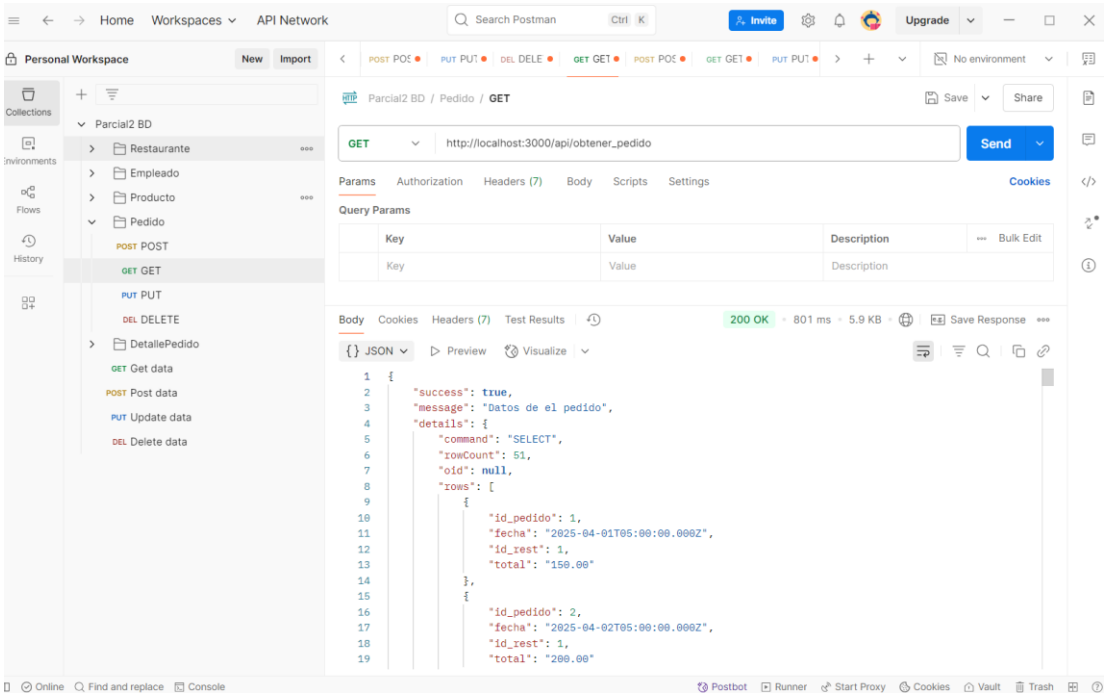
```
452  //MÉTODO DELETE
453
454  app.delete('/api/eliminar_pedido/:id_pedido', (req, res) => {
455      const { id_pedido} = req.params;
456      const query = 'DELETE FROM Pedido WHERE id_pedido = $1';
457
458      connection.query(query, [id_pedido], (error, result) => {
459
460          if (error) {
461              res.status(500).json({
462                  success: false,
463                  message: "Error al eliminar el pedido",
464                  details: error.message
465              });
466          } else if (result.rowCount === 0) {
467              res.status(404).json({
468                  success: false,
469                  message: `No existe el pedido ${id_pedido}`,
470
471              });
472          } else {
473              res.status(200).json({
474                  success: true,
475                  message: "Pedido eliminado de la tabla",
476                  details: result
477
478              });
479          }
480      });
481  });
```

34. Probamos cada API en nuestro Postman de la tabla ***Pedido***.

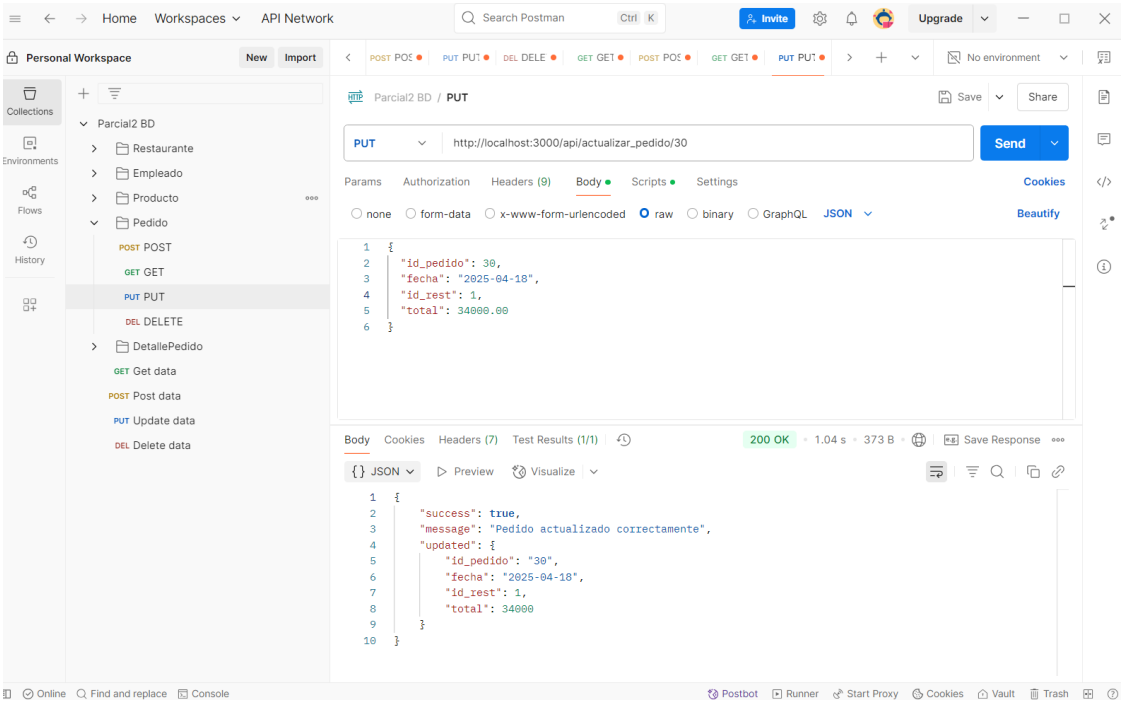
Post


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Personal Workspace
New
Import
No environment
Collections
Environments
Flows
History
Parcial2 BD
Restaurante
Empleado
Producto
Pedido
POST POST
GET GET
PUT PUT
DEL DELETE
DetallePedido
GET Get data
POST Post data
PUT Update data
DEL Delete data
Parcial2 BD / POST
Save
Share
POST
http://localhost:3000/api/guardar_pedido
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"id_pedido": 51,
"fecha": "2025-04-10",
"id_rest": 9,
"total": 280000.00
}
Body
Cookies
Headers (7)
Test Results (1/1)
201 Created
964 ms
304 B
Save Response
JSON
Preview
Visualize
{
"id_pedido": 51,
"fecha": "2025-04-10",
"id_rest": 9,
"total": 280000
}
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

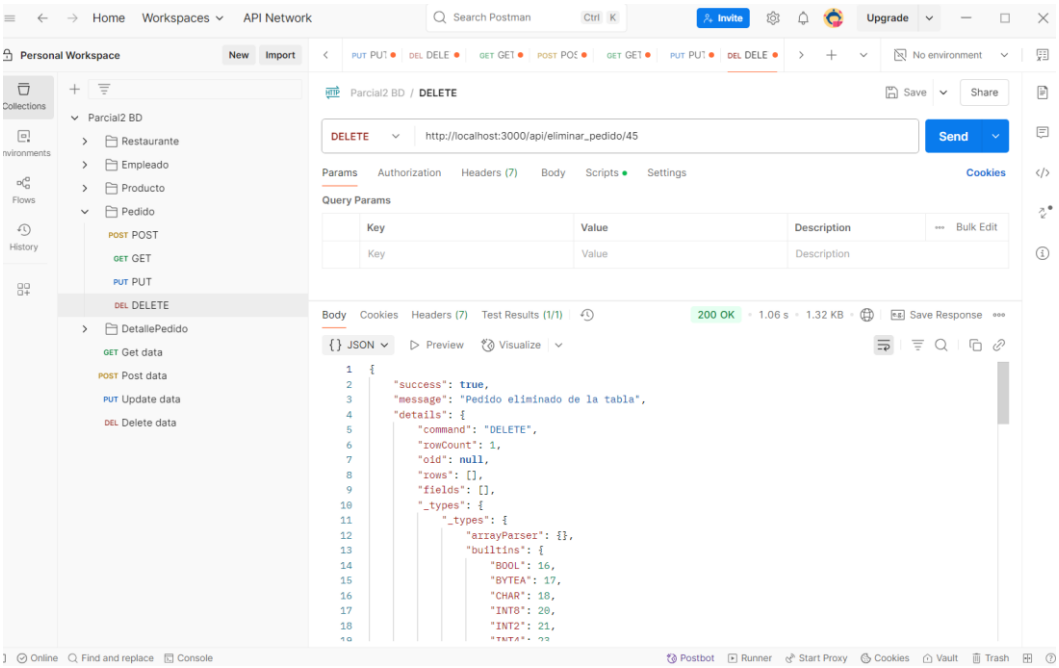
Get


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
Personal Workspace
New
Import
No environment
Collections
Environments
Flows
History
Parcial2 BD
Restaurante
Empleado
Producto
Pedido
POST POST
GET GET
PUT PUT
DEL DELETE
DetallePedido
GET Get data
POST Post data
PUT Update data
DEL Delete data
Parcial2 BD / Pedido / GET
Save
Share
GET
http://localhost:3000/api/obtener_pedido
Send
Params
Authorization
Headers (7)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (7)
Test Results
200 OK
801 ms
5.9 KB
Save Response
JSON
Preview
Visualize
{
"success": true,
"message": "Datos de el pedido",
"details": {
"command": "SELECT",
"rowCount": 51,
"oid": null,
"rows": [
{
"id_pedido": 1,
"fecha": "2025-04-01T05:00:00.000Z",
"id_rest": 1,
"total": "150.00"
},
{
"id_pedido": 2,
"fecha": "2025-04-02T05:00:00.000Z",
"id_rest": 1,
"total": "200.00"
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

Put

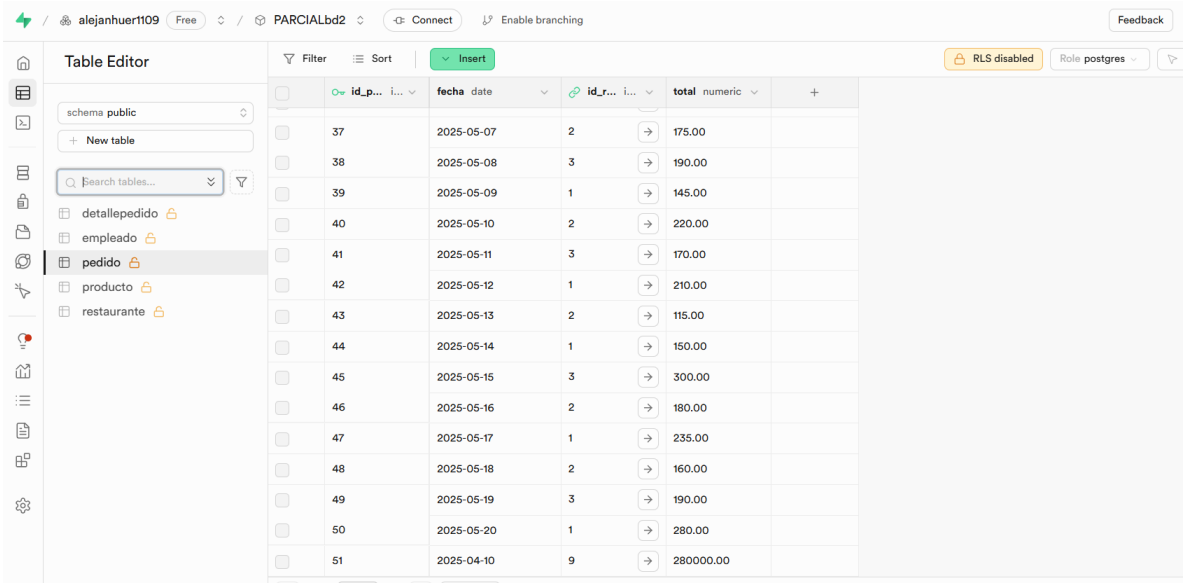


Delete

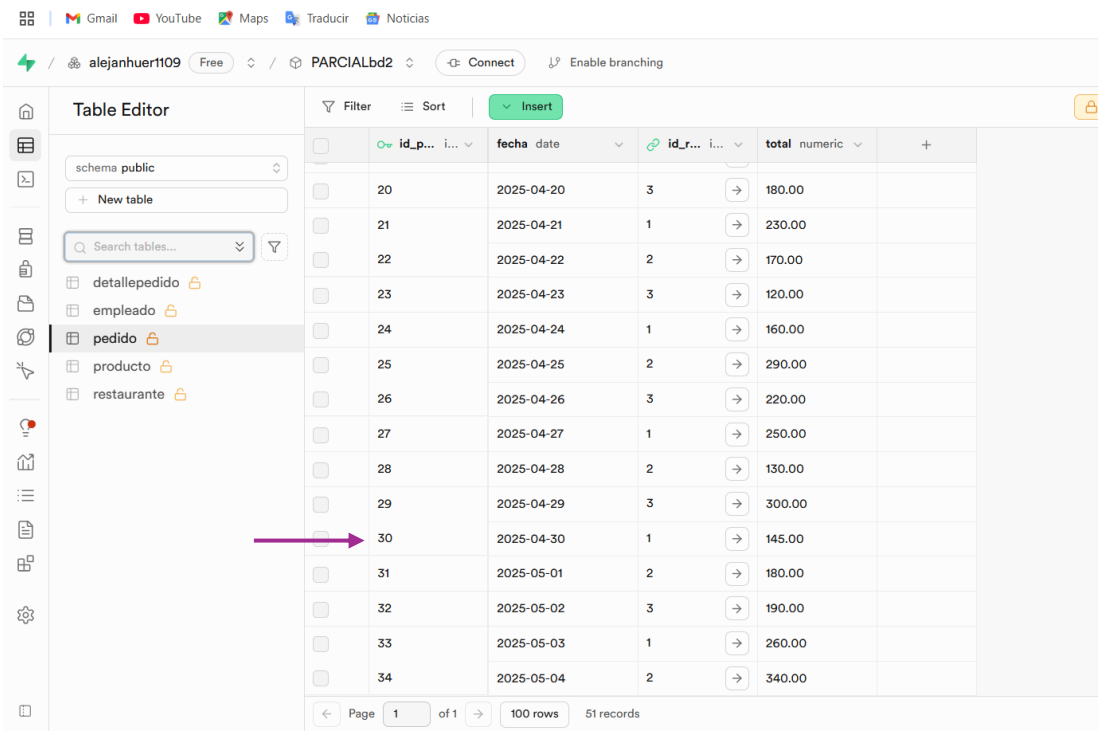


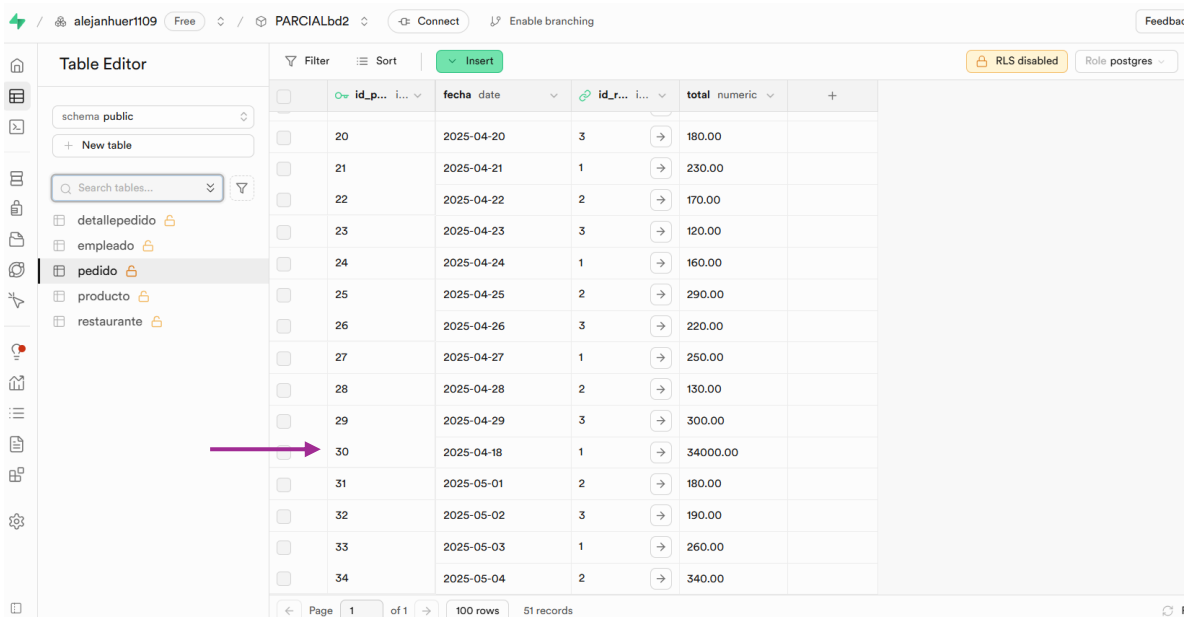
35. Verificamos la actualización de datos de la tabla ***Pedido*** en nuestro Supabase.

Post

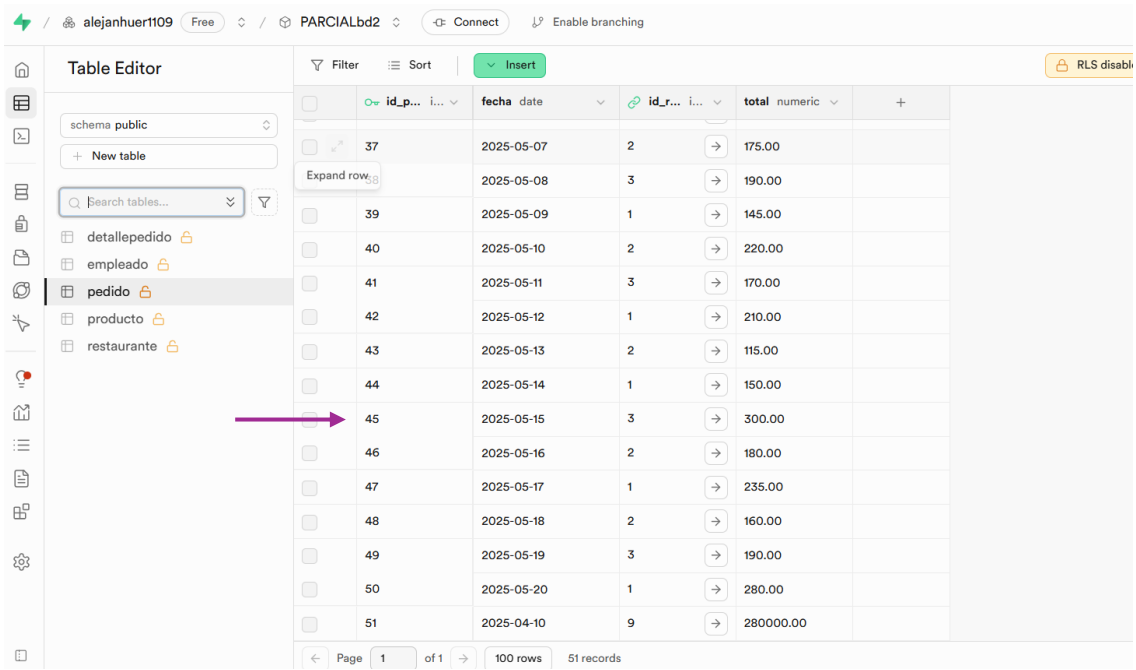

alejanhuer1109
Free
PARCIALbd2
Connect
Enable branching
Feedback
Table Editor
schema public
New table
Search tables...
detallepedido
empleado
pedido
producto
restaurante
Filter
Sort
Insert
RLS disabled
Role postgres
id_p... i...
fecha date
id_r... i...
total numeric
37 2025-05-07 2 175.00
38 2025-05-08 3 190.00
39 2025-05-09 1 145.00
40 2025-05-10 2 220.00
41 2025-05-11 3 170.00
42 2025-05-12 1 210.00
43 2025-05-13 2 115.00
44 2025-05-14 1 150.00
45 2025-05-15 3 300.00
46 2025-05-16 2 180.00
47 2025-05-17 1 235.00
48 2025-05-18 2 160.00
49 2025-05-19 3 190.00
50 2025-05-20 1 280.00
51 2025-04-10 9 280000.00

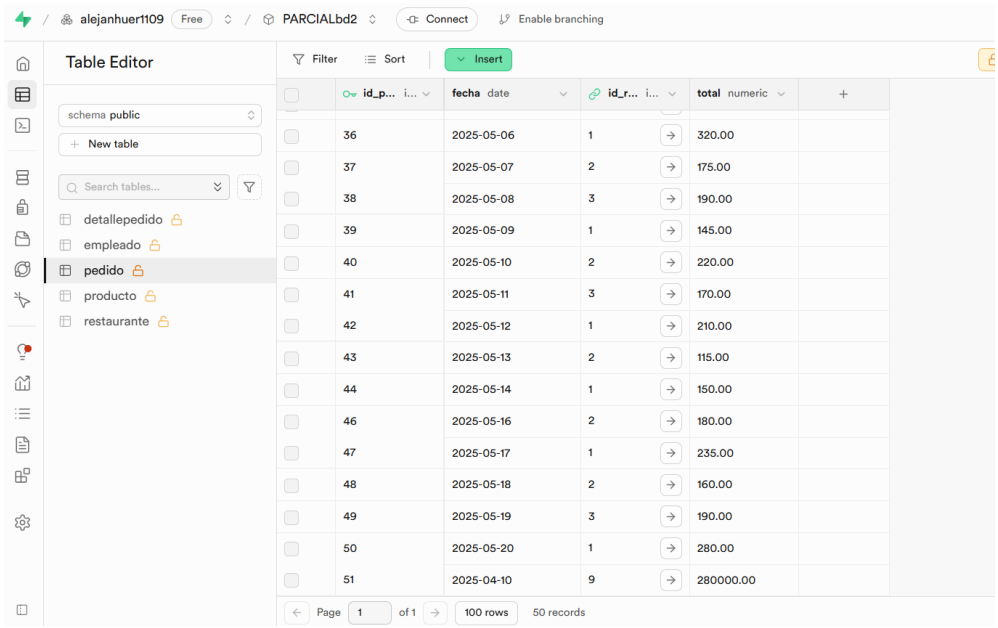
Put


Gmail
YouTube
Maps
Traducir
Noticias
alejanhuer1109
Free
PARCIALbd2
Connect
Enable branching
Table Editor
schema public
New table
Search tables...
detallepedido
empleado
pedido
producto
restaurante
Filter
Sort
Insert
id_p... i...
fecha date
id_r... i...
total numeric
20 2025-04-20 3 180.00
21 2025-04-21 1 230.00
22 2025-04-22 2 170.00
23 2025-04-23 3 120.00
24 2025-04-24 1 160.00
25 2025-04-25 2 290.00
26 2025-04-26 3 220.00
27 2025-04-27 1 250.00
28 2025-04-28 2 130.00
29 2025-04-29 3 300.00
30 2025-04-30 1 145.00
31 2025-05-01 2 180.00
32 2025-05-02 3 190.00
33 2025-05-03 1 260.00
34 2025-05-04 2 340.00
Page 1 of 1
100 rows
51 records


alejanhuer1109
Free
PARCIALbd2
Connect
Enable branching
Table Editor
schema public
New table
Search tables...
detallepedido
empleado
pedido
producto
restaurante
Filter
Sort
Insert
RLS disabled
Role postgres
id_p... i...
fecha date
id_r... i...
total numeric
20 2025-04-20 3 180.00
21 2025-04-21 1 230.00
22 2025-04-22 2 170.00
23 2025-04-23 3 120.00
24 2025-04-24 1 160.00
25 2025-04-25 2 290.00
26 2025-04-26 3 220.00
27 2025-04-27 1 250.00
28 2025-04-28 2 130.00
29 2025-04-29 3 300.00
30 2025-04-18 1 34000.00
31 2025-05-01 2 180.00
32 2025-05-02 3 190.00
33 2025-05-03 1 260.00
34 2025-05-04 2 340.00
Page 1 of 1
100 rows
51 records

Delete


alejanhuer1109
Free
PARCIALbd2
Connect
Enable branching
Table Editor
schema public
New table
Search tables...
detallepedido
empleado
pedido
producto
restaurante
Filter
Sort
Insert
id_p... i...
fecha date
id_r... i...
total numeric
Expand row
37 2025-05-07 2 175.00
2025-05-08 3 190.00
39 2025-05-09 1 145.00
40 2025-05-10 2 220.00
41 2025-05-11 3 170.00
42 2025-05-12 1 210.00
43 2025-05-13 2 115.00
44 2025-05-14 1 150.00
45 2025-05-15 3 300.00
46 2025-05-16 2 180.00
47 2025-05-17 1 235.00
48 2025-05-18 2 160.00
49 2025-05-19 3 190.00
50 2025-05-20 1 280.00
51 2025-04-10 9 280000.00
Page 1 of 1
100 rows
51 records



36. Se crea el CRUD para la tabla ***DetallePedido*** en nuestro archivo index.js.

Post

```
//CRUD DETALLE PEDIDO

//MÉTODO POST

app.post('/api/guardar_DetallePedido', (req, res) => {
    const { id_detalle, id_pedido, id_prod, cantidad, subtotal } = req.body;
    const query = 'INSERT INTO DetallePedido (id_detalle, id_pedido, id_prod, cantidad, subtotal) VALUES ($1, $2, $3, $4, $5)';

    connection.query(query, [id_detalle, id_pedido, id_prod, cantidad, subtotal], (error, result) => {
        if (error) {
            res.status(500).json({
                message: 'ERROR CREANDO EL DETALLE DEL PEDIDO',
                error
            });
        } else {
            res.status(201).json({
                success: true,
                message: 'Detalle del pedido creado correctamente',
                created: { id_detalle, id_pedido, id_prod, cantidad, subtotal }
            });
        }
    });
});

```

Get

```
//MÉTODO GET

app.get('/api/obtener_DetallePedido', (req, res) => {
    const query = 'SELECT * FROM DetallePedido';

    connection.query(query, (error, result) => {
        if (error) {
            res.status(500).json({
                success: false,
                message: "Error al recuperar los datos de el detella del pedido",
                details: error.message
            });
        } else {
            res.status(200).json({
                success: true,
                message: "Detalles de el pedido",
                details: result
            });
        }
    });
});
```

Put

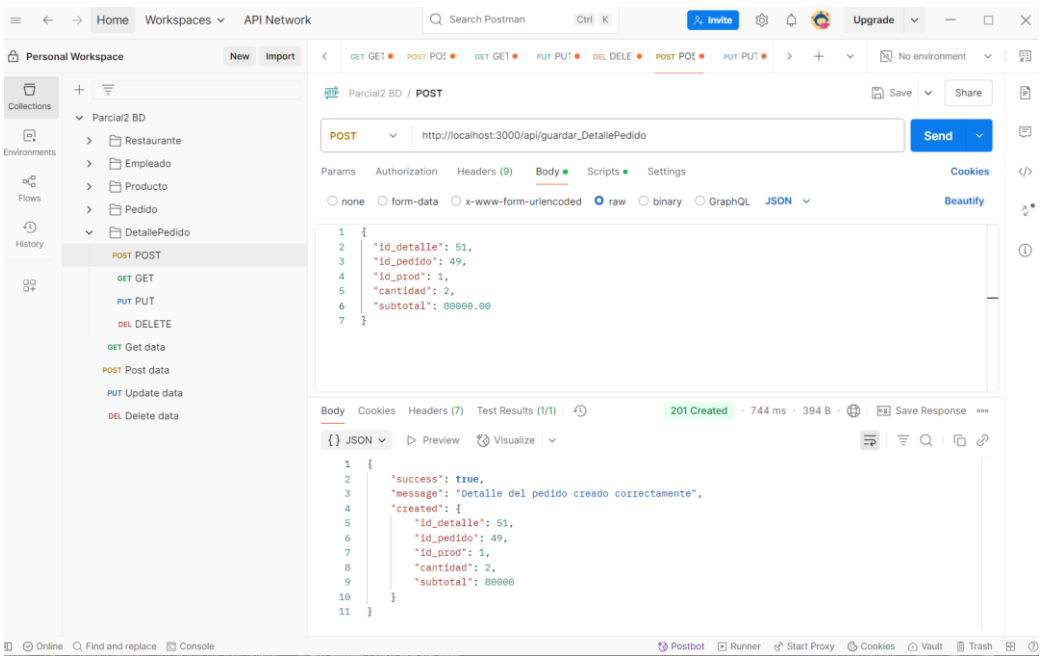
```
//METODO PUT
app.put('/api/actualizar_DetallePedido/:id_detalle', (req, res) => {
    const { id_detalle } = req.params;
    const { id_pedido, id_prod, cantidad, subtotal} = req.body;

    const query = 'UPDATE DetallePedido SET id_pedido = $1, id_prod = $2, cantidad = $3, subtotal = $4 WHERE id_detalle = $5';

    connection.query(query, [id_pedido, id_prod, cantidad, subtotal, id_detalle], (error, result) => {
        if (error) {
            res.status(500).json({
                success: false,
                message: 'Error al actualizar el detalle del pedido',
                details: error.message
            });
        } else if (result.rowCount === 0) {
            res.status(404).json({
                success: false,
                message: `No se encontró ningún detalle de pedido con el id ${id_detalle}`
            });
        } else {
            res.status(200).json({
                success: true,
                message: 'Detalle de pedido actualizado correctamente',
                updated: {
                    id_detalle,
                    id_pedido,
                    id_prod,
                    cantidad,
                    subtotal
                }
            });
        }
    });
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS
```

Delete

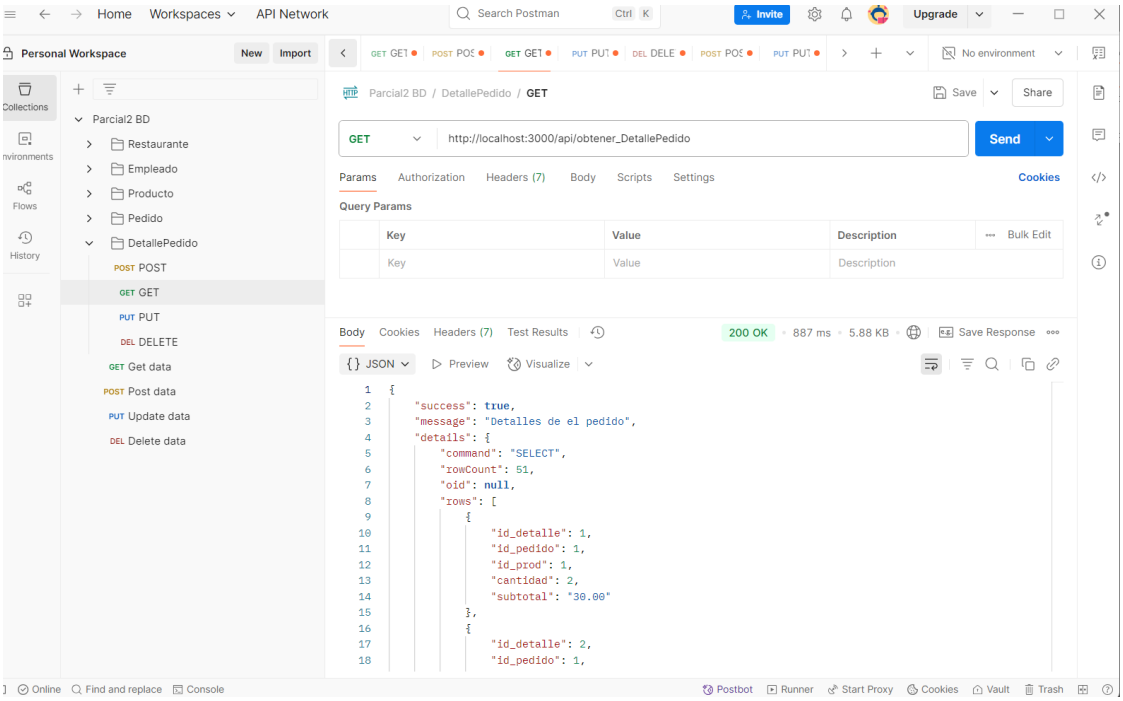
```
568  //MÉTODO DELETE
569
570  app.delete('/api/eliminar_DetallePedido/:id_detalle', (req, res) => {
571      const { id_detalle} = req.params;
572      const query = 'DELETE FROM DetallePedido WHERE id_detalle = $1';
573
574      connection.query(query, [id_detalle], (error, result) => {
575
576          if (error) {
577              res.status(500).json({
578                  success: false,
579                  message: "Error al eliminar el detalle del pedido",
580                  details: error.message
581              });
582          } else if (result.rowCount === 0) {
583              res.status(404).json({
584                  success: false,
585                  message: `No existe el detalle del pedido ${id_detalle}`,
586
587              });
588          } else {
589              res.status(200).json({
590                  success: true,
591                  message: "Detalle del pedido eliminado de la tabla",
592                  details: result
593
594              });
595          }
596      });
597  });
```

37. Probamos cada API en nuestro Postman de la tabla ***DetallePedido***.

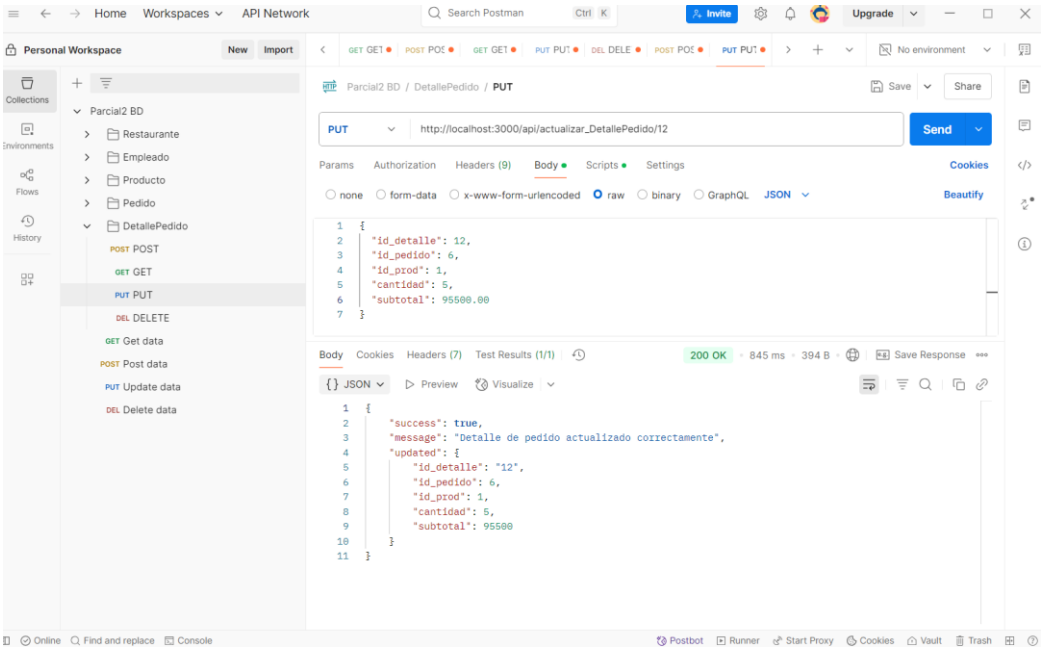
Post



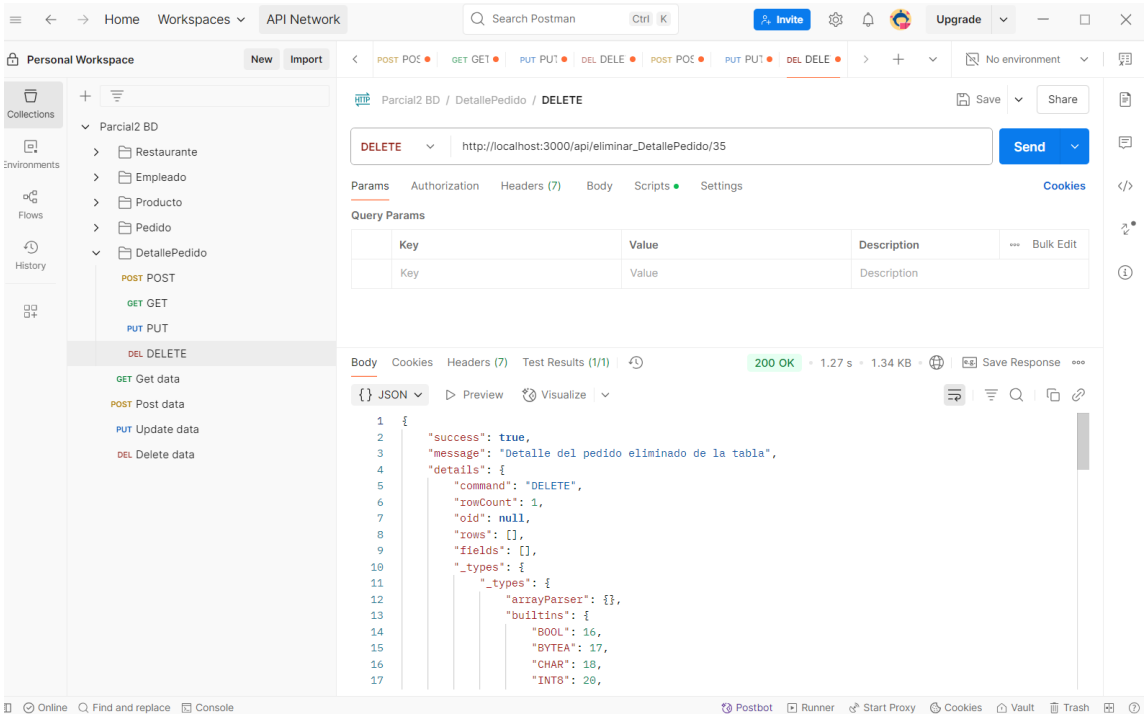
Get



Put

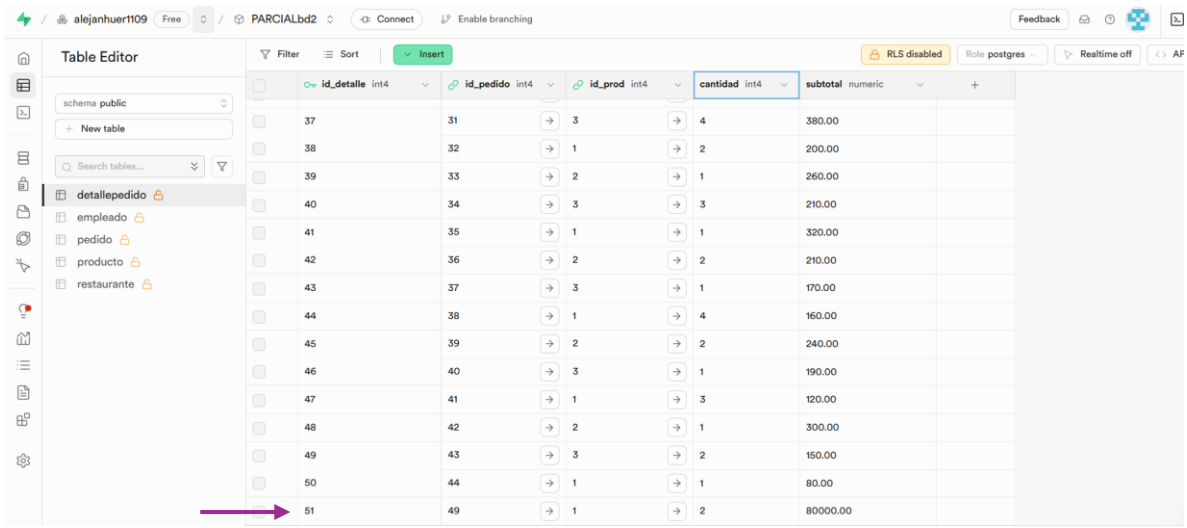


Delete

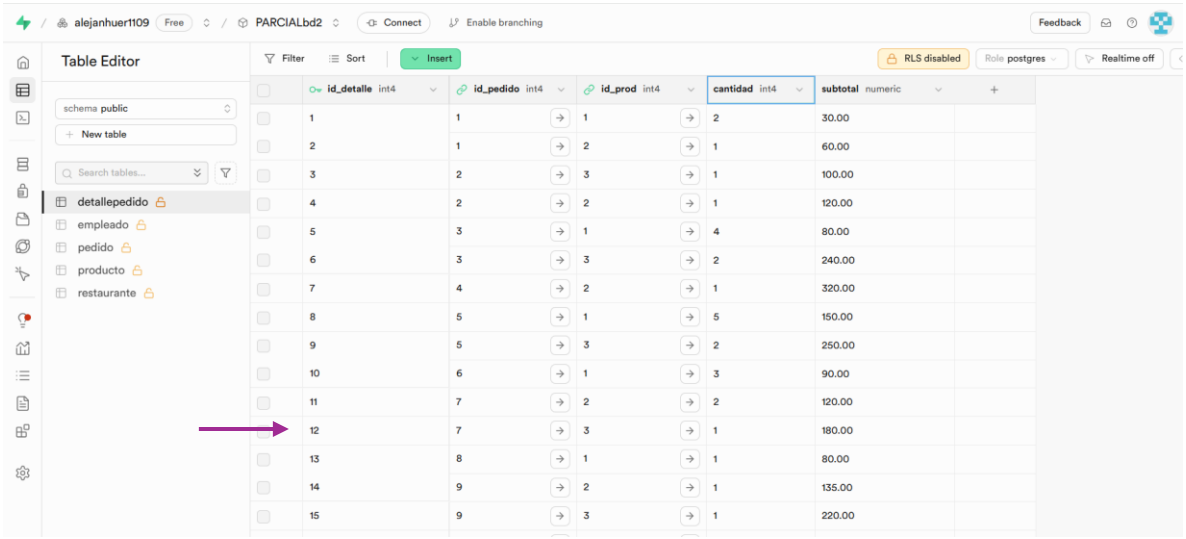


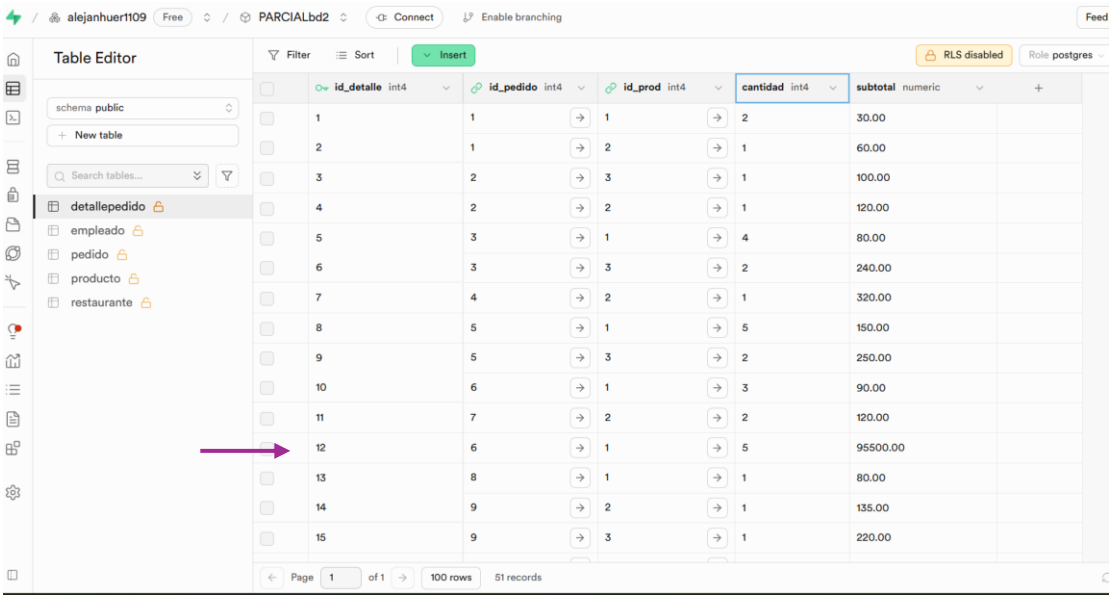
38. Verificamos la actualización de datos de la tabla ***DetallePedido*** en nuestro Supabase.

Post

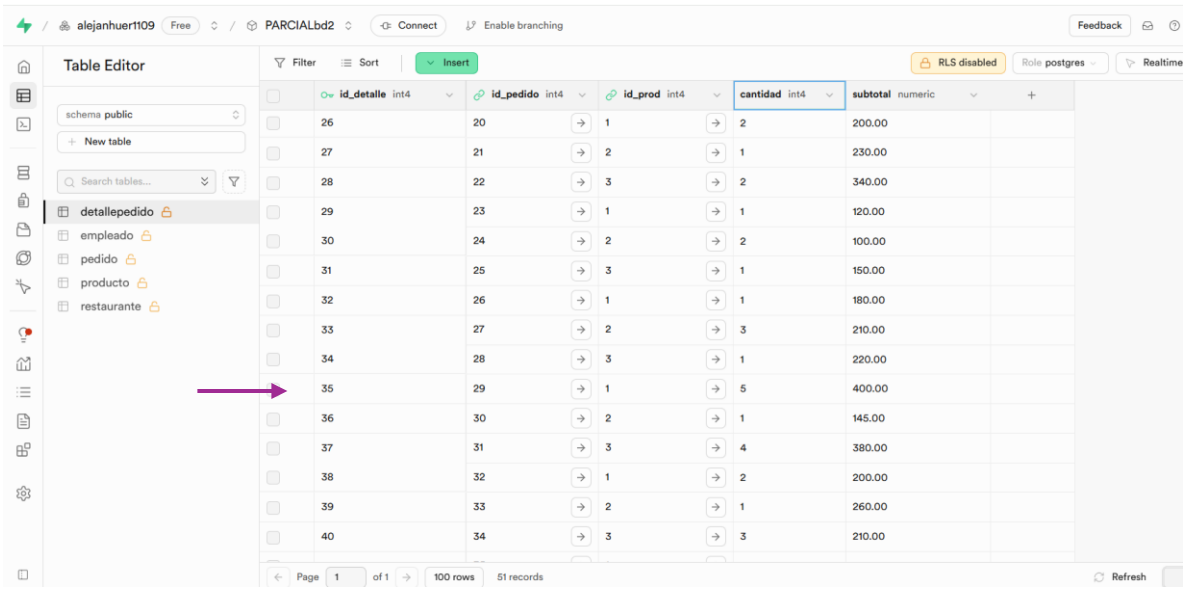


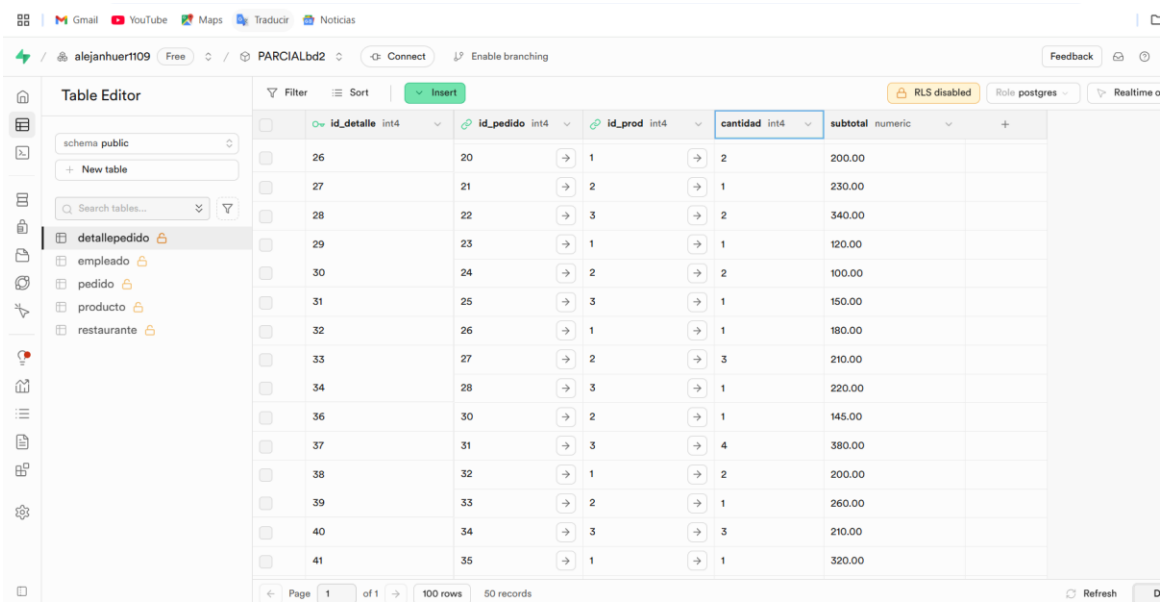
Put


alejanhuer1109
Free
PARCIALbd2
Connect
Enable branching
Feedback
Table Editor
Filter
Sort
Insert
RLS disabled
Role postgres
Realtime off
schema public
New table
Search tables...
detallepedido
empleado
pedido
producto
restaurante
id_detalle int4
id_pedido int4
id_prod int4
cantidad int4
subtotal numeric
1 1 1 2 30.00
2 1 2 1 60.00
3 2 3 1 100.00
4 2 2 1 120.00
5 3 1 4 80.00
6 3 3 2 240.00
7 4 2 1 320.00
8 5 1 5 150.00
9 5 3 2 250.00
10 6 1 3 90.00
11 7 2 2 120.00
12 7 3 1 180.00
13 8 1 1 80.00
14 9 2 1 135.00
15 9 3 1 220.00


alejanhuer1109
Free
PARCIALbd2
Connect
Enable branching
Table Editor
Filter
Sort
Insert
RLS disabled
Role postgres
schema public
New table
Search tables...
detallepedido
empleado
pedido
producto
restaurante
id_detalle int4
id_pedido int4
id_prod int4
cantidad int4
subtotal numeric
1 1 1 2 30.00
2 1 2 1 60.00
3 2 3 1 100.00
4 2 2 1 120.00
5 3 1 4 80.00
6 3 3 2 240.00
7 4 2 1 320.00
8 5 1 5 150.00
9 5 3 2 250.00
10 6 1 3 90.00
11 7 2 2 120.00
12 6 1 5 95500.00
13 8 1 1 80.00
14 9 2 1 135.00
15 9 3 1 220.00
Page 1 of 1
100 rows
51 records

Delete





39. Realizar la codificación de las consultas nativas en el archivo index.js y se ejecutan desde Postman.

1. Obtener todos los productos de un pedido específico

Codificación en Visual Studio Code

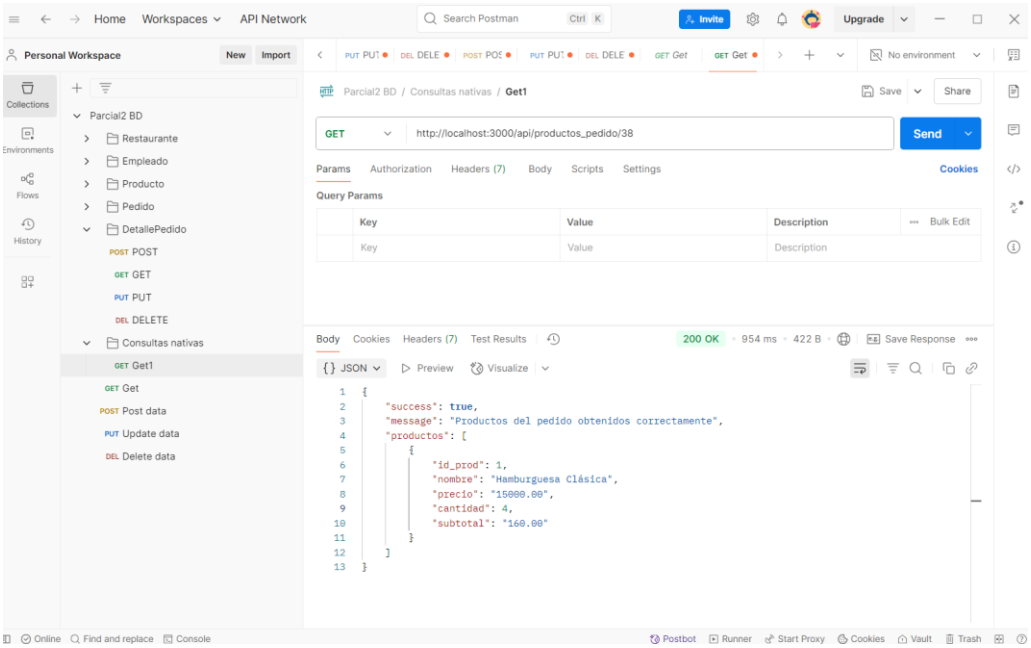
A través del código se obtienen todos los productos que están asociados a un pedido específico (por su id_pedido), incluyendo información detallada

de cada producto dentro del pedido. Por lo tanto devuelve una lista con los productos de ese pedido, mostrando:

- **id_prod:** ID del producto
- **nombre:** Nombre del producto
- **precio:** Precio del producto
- **cantidad:** Cuántas unidades se pidieron de ese producto
- **subtotal:** Precio total por esas unidades (cantidad × precio)

```
599  //CONSULTAS NATIVAS
600
601  // 1. Obtener todos los productos de un pedido específico
602  app.get('/api/productos_pedido/:id_pedido', (req, res) => {
603      const { id_pedido } = req.params;
604      const query = `
605          SELECT P.id_prod, P.nombre, P.precio, DP.cantidad, DP.subtotal
606          FROM DetallePedido DP
607          JOIN Producto P ON DP.id_prod = P.id_prod
608          WHERE DP.id_pedido = $1
609      `;
610
611      connection.query(query, [id_pedido], (error, result) => {
612          if (error) {
613              res.status(500).json({
614                  success: false,
615                  message: "Error al obtener los productos del pedido",
616                  details: error.message
617              });
618          } else {
619              res.status(200).json({
620                  success: true,
621                  message: "Productos del pedido obtenidos correctamente",
622                  productos: result.rows
623              });
624          }
625      });
626  });
627
```

Validación en Postman



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

Codificación en Visual Studio Code

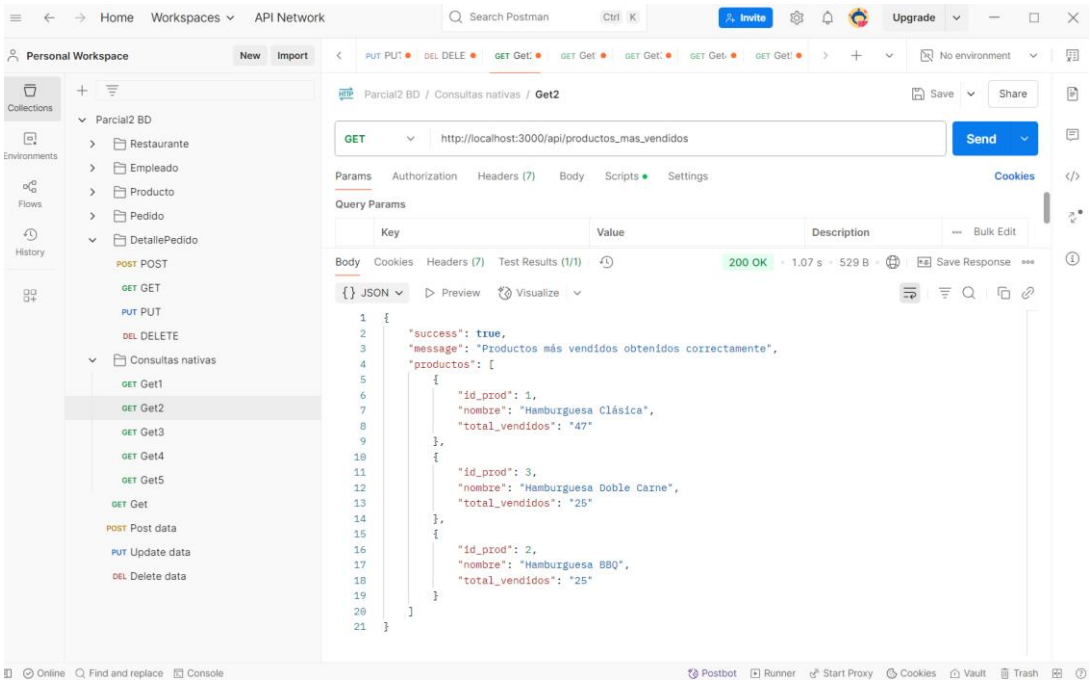
Esta consulta une las tablas:

- DetallePedido (donde se registra cuántas unidades de cada producto se pidieron)
- Producto (para obtener el nombre y el id del producto)

Luego agrupa por id_prod y nombre para obtener datos únicos por producto, suma la cantidad total vendida por producto con SUM(DP.cantidad), ordena los resultados de mayor a menor (ORDER BY total_vendidos DESC) y limita los resultados a los 10 primeros con LIMIT 10.

```
628  //Obtener los productos más vendidos (más de X unidades)
629
630  app.get('/api/productos_mas_vendidos', (req, res) => {
631      const query = `
632          SELECT P.id_prod, P.nombre, SUM(DP.cantidad) AS total_vendidos
633          FROM DetallePedido DP
634          JOIN Producto P ON DP.id_prod = P.id_prod
635          GROUP BY P.id_prod, P.nombre
636          ORDER BY total_vendidos DESC
637          LIMIT 10
638      `;
639
640      connection.query(query, (error, result) => {
641          if (error) {
642              res.status(500).json({
643                  success: false,
644                  message: "Error al obtener los productos más vendidos",
645                  details: error.message
646              });
647          } else {
648              res.status(200).json({
649                  success: true,
650                  message: "Productos más vendidos obtenidos correctamente",
651                  productos: result.rows
652              });
653          }
654      });
655  });
656
```

Validación en Postman



3. Obtener el total de ventas por restaurante

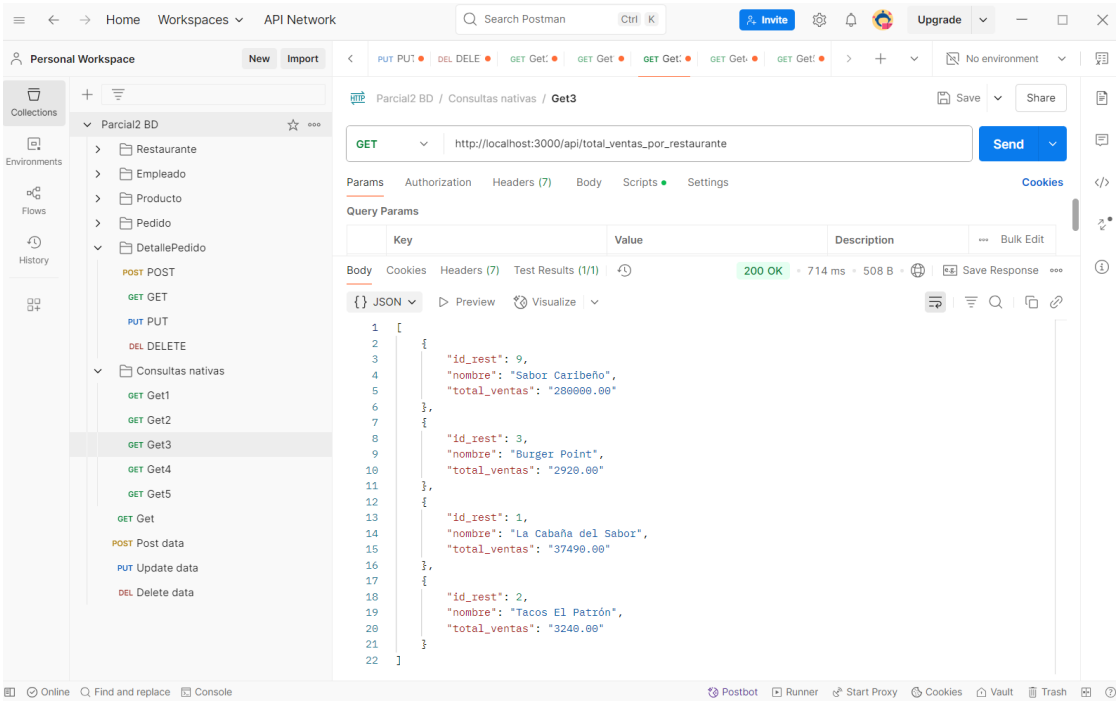
Codificación en Visual Studio Code

- Restaurante (r): para obtener el nombre e ID del restaurante.
- Pedido (p): que contiene el campo total, el monto total de cada pedido.

En el código se relacionan ambas tablas con un JOIN usando r.id_rest = p.id_rest, luego se agrupa los pedidos por restaurante con GROUP BY r.id_rest, r.nombre y se suman todos los valores del campo total de los pedidos de cada restaurante con SUM(p.total).

```
657  //Obtener el total de ventas por restaurante
658
659  app.get('/api/total_ventas_por_restaurante', (req, res) => {
660      const query = `
661          SELECT r.id_rest, r.nombre, SUM(p.total) AS total_ventas
662          FROM Restaurante r
663          JOIN Pedido p ON r.id_rest = p.id_rest
664          GROUP BY r.id_rest, r.nombre
665      `;
666
667      connection.query(query, (error, result) => {
668          if (error) {
669              res.status(500).json({ message: "Error al obtener el total de ventas", error });
670          } else {
671              res.status(200).json(result.rows);
672          }
673      });
674  });
675
```

Validación en Postman



4. Obtener los pedidos realizados en una fecha específica

Codificación en Visual Studio Code

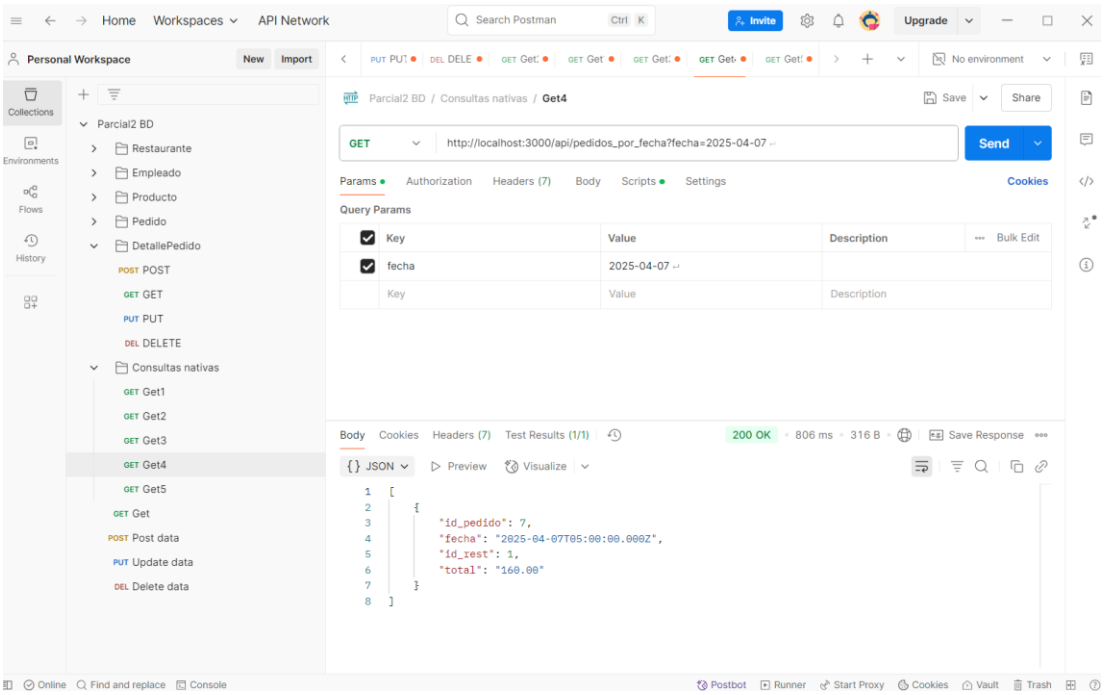
El código recibe un parámetro fecha por query string, luego ejecuta la consulta SQL ***SELECT * FROM Pedido WHERE fecha = $1.***

Después reemplaza $1 con la fecha recibida y busca todos los registros en la tabla Pedido cuya fecha coincida exactamente con la proporcionada.

Para finalizar devuelve los pedidos encontrados en formato JSON o un error si ocurre algo.

```
676  //4. Obtener los pedidos realizados en una fecha específica
677
678  app.get('/api/pedidos_por_fecha', (req, res) => {
679      const { fecha } = req.query;
680      const query = `SELECT * FROM Pedido WHERE fecha = $1`;
681
682      connection.query(query, [fecha], (error, result) => {
683          if (error) {
684              res.status(500).json({ message: "Error al obtener pedidos por fecha", error });
685          } else {
686              res.status(200).json(result.rows);
687          }
688      });
689  });
```

Validación en Postman



5. Obtener los empleados por rol en un restaurante

Codificación en Visual Studio Code

El código recibe dos parámetros por query string:

- **rol:** el rol que quieres filtrar (por ejemplo: "mesero", "cocinero").
- **id_rest:** el ID del restaurante.

Luego ejecuta la consulta ***SELECT * FROM Empleado WHERE rol = $1 AND id_rest = $2*** donde $1 se reemplaza por el valor de rol y $2 por el valor de id_rest.

Para finalizar devuelve un arreglo de empleados que cumplen con esos filtros o un error si algo sale mal.

```
691  //5. Obtener los empleados por rol en un restaurante
692
693  app.get('/api/empleados_por_rol', (req, res) => {
694      const { rol, id_rest } = req.query;
695      const query = `SELECT * FROM Empleado WHERE rol = $1 AND id_rest = $2`;
696
697      connection.query(query, [rol, id_rest], (error, result) => {
698          if (error) {
699              res.status(500).json({ message: "Error al obtener empleados por rol", error });
700          } else {
701              res.status(200).json(result.rows);
702          }
703      });
704  });
705
```

Validación en Postman