

## **Parcial 2**

Paula Alejandra Martínez Huertas

Ingeniería de sistemas, Corporación Universitaria Minuto de Dios

NRC 60747: Base de datos masivas

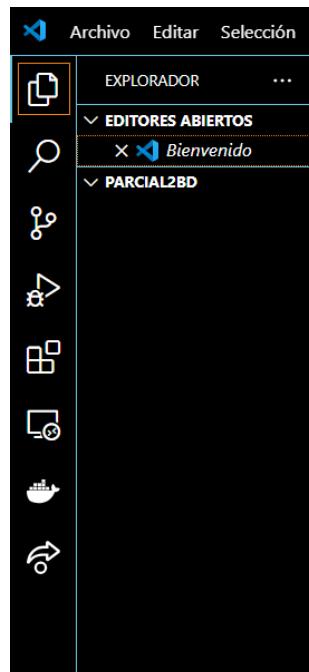
Ing. William Matallana Porras

Abril 25, 2025

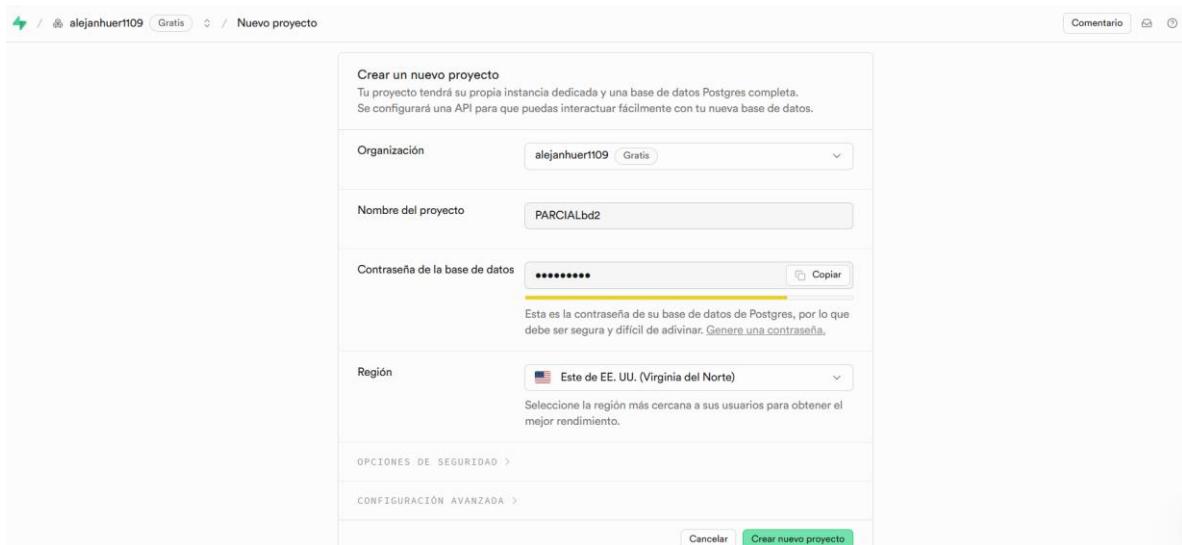
1. Creamos una carpeta en el escritorio, en este caso llamada “**“Parcial2BD”**”.



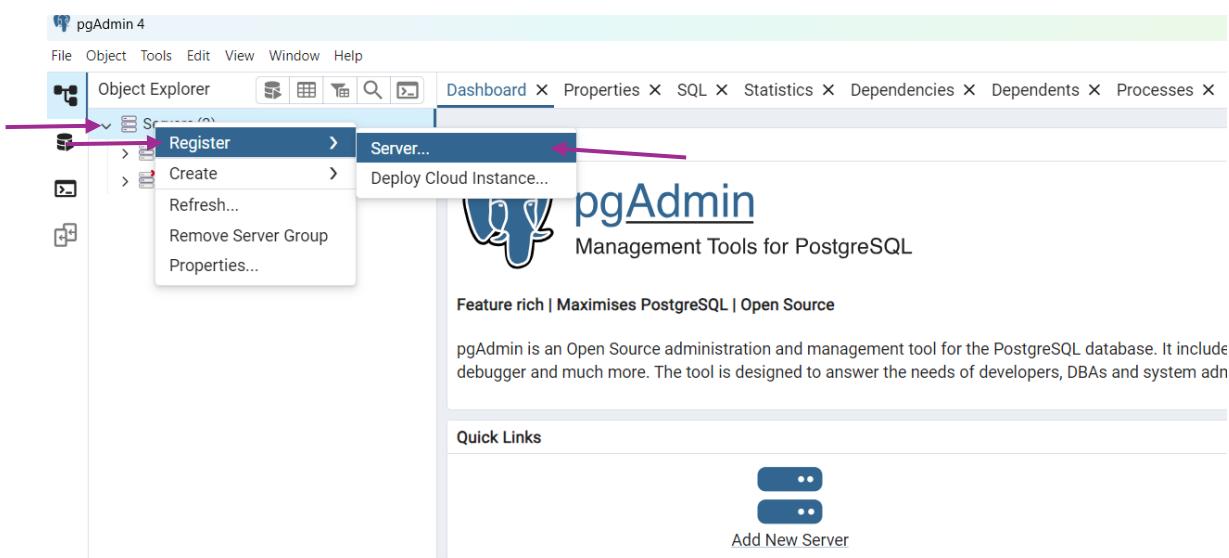
2. Abrimos la carpeta creada en el espacio de trabajo de nuestro Visual Studio Code.



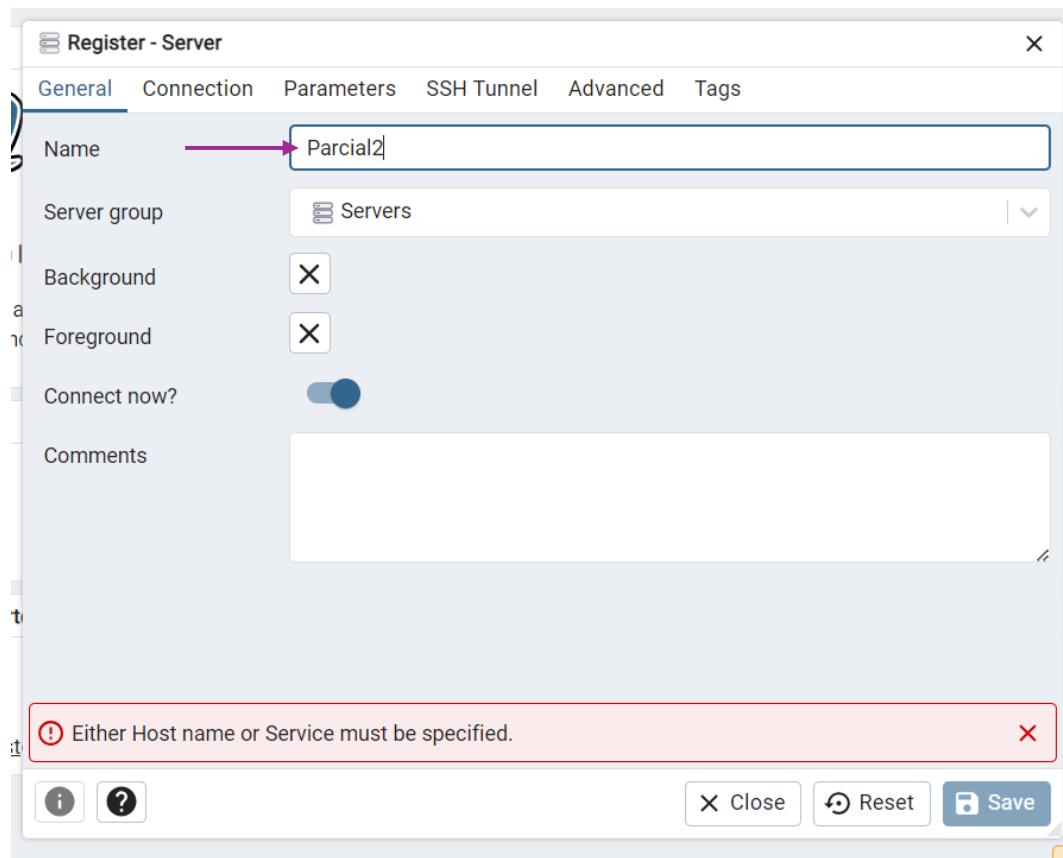
3. Creamos un nuevo proyecto en Supabase, en este caso llamado “**“PARCIALbd2”**, se establece la contraseña de la base de datos y se da clic en la opción Crear nuevo proyecto.



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]@  
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**

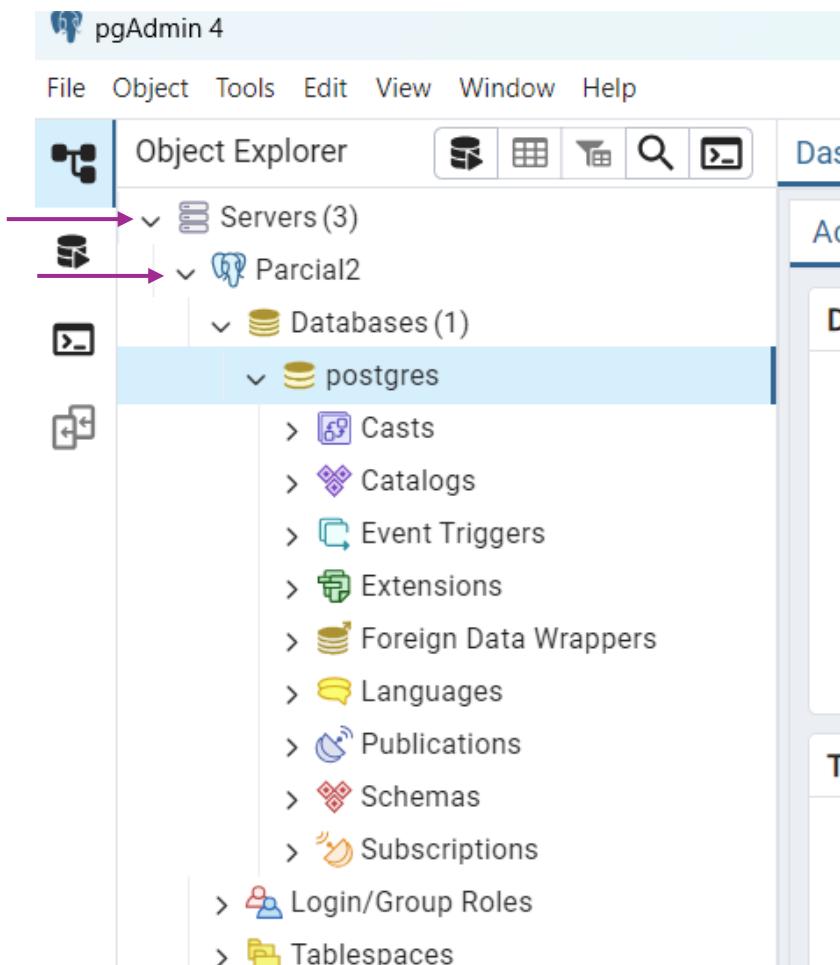
**Connection**

Host name/address	aws-0-us-east-1.pooler.supabase.com
Port	5432
Maintenance database	postgres
Username	postgres.uioactaqycnvxnamgmi
Kerberos authentication?	<input type="checkbox"/>
Password	*****
Save password?	<input checked="" type="checkbox"/>
Role	
Service	

**General Parameters SSH Tunnel Advanced Tags**

**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 lateral 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

**pgAdmin 4**

File Object Tools Edit View Window Help

Object Explorer    Dashboard Properties SQL Statistics Dependencies Dependents Processes postgres/postgres@Parcial2\*

Servers (3) Parcial2 Databases (1) postgres

- Casts
- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Publications
- Schemas
- Subscriptions
- Login/Group Roles
- Tablespaces

PostgreSQL 17 supabaseBD

Query Query History

```

1 v CREATE TABLE Restaurante (
2   id_rest SERIAL PRIMARY KEY,
3   nombre VARCHAR(100),
4   ciudad VARCHAR(100),
5   direccion VARCHAR(150),
6   fecha_apertura DATE
7 );
8

```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 375 msec.

✓ Query returned successfully in 375 msec. X

## Tabla Empleado

**pgAdmin 4**

File Object Tools Edit View Window Help

Object Explorer    Dashboard Properties SQL Statistics Dependencies Dependents Processes postgres/postgres@Parcial2\*

Servers (3) Parcial2 Databases (1) postgres

- Casts
- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Publications
- Schemas
- Subscriptions
- Login/Group Roles
- Tablespaces

PostgreSQL 17 supabaseBD

Query Query History

```

1 v CREATE TABLE Empleado (
2   id_empleado SERIAL PRIMARY KEY,
3   nombre VARCHAR(100),
4   rol VARCHAR(50),
5   id_rest INT REFERENCES Restaurante(id_rest)
6 );
7

```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 217 msec.

Total rows: Query complete 00:00:00.217

✓ Query returned successfully in 217 msec. X

CRLF Ln 7, Col 1

## Tabla Producto

**Tabla Producto**

The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'postres' database in the 'Parcial2' server, the 'Tables' node is selected. In the main query editor, the following SQL code is written:

```

1 v CREATE TABLE Producto (
2   id_prod SERIAL PRIMARY KEY,
3   nombre VARCHAR(100),
4   precio NUMERIC(10,2)
5 );
6

```

The 'Messages' tab at the bottom shows the execution results:

```

CREATE TABLE
Query returned successfully in 174 msec.

Total rows: Query complete 00:00:00.174

```

A green status bar at the bottom right indicates: **✓ Query returned successfully in 174 msec. CRLF Ln 6, Col 1**.

## Tabla Pedido

The screenshot shows the pgAdmin 4 interface. In the Object Explorer, under the 'postres' database in the 'Parcial2' server, the 'Tables' node is selected. In the main query editor, the following SQL code is written:

```

1 v CREATE TABLE Pedido (
2   id_pedido SERIAL PRIMARY KEY,
3   fecha DATE,
4   id_rest INT REFERENCES Restaurante(id_rest),
5   total NUMERIC(10,2)
6 );
7

```

The 'Messages' tab at the bottom shows the execution results:

```

CREATE TABLE
Query returned successfully in 189 msec.

Total rows: Query complete 00:00:00.189

```

A green status bar at the bottom right indicates: **✓ Query returned successfully in 189 msec. CRLF Ln 7, Col 1**.

## Tabla DetallePedido

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer    Dashboard Properties SQL Statistics Dependencies Dependents Processes postgres/postgres.luiactagycnvnamgmi@Parcial2

Servers (3)    Databases (1)

PostgreSQL    Databases (1)

postgres    Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Publications Schemas Subscriptions Login/Group Roles Tablespaces PostgreSQL 17 supabaseBD

Query History    Scratch Pad

```

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

```

Data Output Messages Notifications

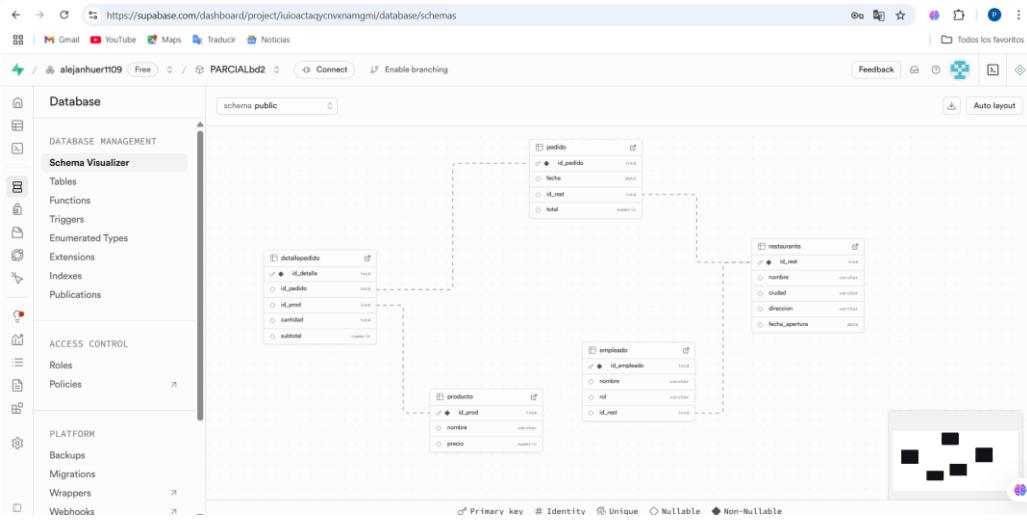
CREATE TABLE

Query returned successfully in 197 msec.

Total rows: Query complete 00:00:00.197 ✓ Query returned successfully in 197 msec. CRLF Ln 8, Col 1

Aggregates Collations Domains FTS Configurations FTS Dictionaries FTS Parsers FTS Templates Foreign Tables Functions Materialized Views Operators Procedures Sequences Tables (5) detallepedido empleado pedido producto restaurante Trigger Functions Types Views

## 9. Modelo relacional obtenido desde Supabase.



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

Name	Description	Rows (Estimated)	Size (Estimated)	Realtime Enabled	Columns
detallepedido	No description	0	8132 bytes	X	5 columns
empleado	No description	0	8132 bytes	X	4 columns
pedido	No description	0	8132 bytes	X	4 columns
producto	No description	0	8132 bytes	X	3 columns
restaurante	No description	0	8132 bytes	X	5 columns

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

### Tabla restaurante

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

- graphql\_public
- pgbouncer
- public
  - Aggregates
  - Collations
  - Domains
  - FTS Configurations
  - FTS Dictionaries
  - FTS Parsers
  - FTS Templates
  - Foreign Tables
  - Functions
  - Materialized Views
  - Operators
  - Procedures
  - Sequences
  - Tables(5)
    - detallepedido
    - empleado
    - pedido
    - producto
    - restaurante
  - Columns(5)
    - id\_rest
    - nombre
    - ciudad
    - direccion
    - fecha\_apertura
  - Constraints
  - Indexes

Dashboard X Properties X SQL X Statistics X Dependencies X Dependents X Processes X postgres/postgres@Parcial2+ X

Query History

```
1. INSERT INTO restaurante (id_rest, nombre, ciudad, direccion, fecha_apertura) VALUES
2. (1, 'La Cabaña del Sabor', 'Bogotá', 'Calle 45 #13-30', '2020-01-15'),
3. (2, 'Tacos El Patrón', 'Medellín', 'Carrera 50 #20-12', '2021-03-10'),
4. (3, 'Burger Point', 'Cali', 'Av 3N #55-89', '2019-06-01'),
5. (4, 'Sazón Express', 'Barranquilla', 'Cra 45 #85-60', '2022-07-22'),
6. (5, 'Pizza Rápida', 'Cartagena', 'Calle del Arsenal #6-21', '2020-09-15'),
7. (6, 'El Corralito', 'Manizales', 'Av Santander #40-22', '2021-05-03'),
8. (7, 'Rápido y Sabroso', 'Pereira', 'Cra 7 #15-33', '2022-02-18'),
9. (8, 'Arepas Gourmet', 'Bucaramanga', 'Calle 36 #27-50', '2019-08-11'),
10. (9, 'Sabor Caribeño', 'Santa Marta', 'Cra 2 #17-89', '2021-12-05'),
11. (10, 'Punto Burgers', 'Ibagué', 'Calle 60 #7-45', '2020-11-05'),
12. (11, 'Delicias del Valle', 'Cali', 'Cra 1 #23-45', '2021-04-10'),
13. (12, 'Tropibites', 'Barrancabermeja', 'Calle 48 #12-20', '2020-08-19'),
14. (13, 'El Buen Gusto', 'Villavicencio', 'Av 40 #26-30', '2021-09-30'),
15. (14, 'Sazón Llanero', 'Yopal', 'Cra 15 #20-10', '2020-12-12'),
16. (15, 'Tostadas & Más', 'Neiva', 'Calle 15 #8-50', '2019-10-07'),
17. (16, 'La Parrilla Rápida', 'Popayán', 'Cra 9 #24-66', '2021-06-18'),
18. (17, 'Fritanga Urbana', 'Pasto', 'Calle 2A #5-30', '2022-01-22'),
19. (18, 'Comidas Don Leo', 'Cúcuta', 'Av 5 #10-90', '2019-11-03'),
20. (19, 'Sabrositos', 'Sincelejo', 'Cra 8 #33-25', '2020-03-17'),
21. (20, 'FastFood Max', 'Montería', 'Calle 29 #6-60', '2021-02-14'),
```

Data Output Messages Notifications

INSERT 0 50

Query returned successfully in 121 msec.

Total rows: 21 Query complete 00:00:00.121

Query return CRLF Ln 1, Col 14

https://supabase.com/dashboard/project/uioactaqcnxnamgmi/editor/17249/schema=public

alejanhuert109 Free / PARCIALbd2 Connect Enable branching

Table Editor

	id_rest	nombre	ciudad	direccion	fecha_apertura
1	La Cabaña del Sabor	Bogotá	Calle 45 #13-30	2020-01-15	
2	Tacos El Patrón	Medellín	Carrera 50 #20-12	2021-03-10	
3	Burger Point	Cali	Av 3N #55-89	2019-06-01	
4	Sazón Express	Barranquilla	Cra 45 #85-60	2022-07-22	
5	Pizza Rápida	Cartagena	Calle del Arsenal #6-21	2020-09-15	
6	El Corralito	Manizales	Av Santander #40-22	2021-05-03	
7	Rápido y Sabroso	Pereira	Cra 7 #15-33	2022-02-18	
8	Arepas Gourmet	Bucaramanga	Calle 36 #27-50	2019-08-11	
9	Sabor Caribeño	Santa Marta	Cra 2 #17-89	2021-12-05	
10	Punto Burgers	Ibagué	Calle 60 #7-45	2020-11-05	
11	Delicias del Valle	Cali	Cra 1 #23-45	2021-04-10	
12	Tropibites	Barrancabermeja	Calle 48 #12-20	2020-08-19	
13	El Buen Gusto	Villavicencio	Av 40 #26-30	2021-09-30	
14	Sazón Llanero	Yopal	Cra 15 #20-10	2020-12-12	
15	Tostadas & Más	Neiva	Calle 15 #8-50	2019-10-07	

Page 1 of 1 100 rows 50 records Refresh Data Default

## Tabla empleado

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

- > **public**
  - > **Aggregates**
  - > **Collations**
  - > **Domains**
  - > **FTS Configurations**
  - > **FTS Dictionaries**
  - > **FTS Parsers**
  - > **FTS Templates**
  - > **Foreign Tables**
  - > **Functions**
  - > **Materialized Views**
  - > **Operators**
  - > **Procedures**
  - > **Sequences**
  - Tables (5)**
    - > **detallepedido**
    - > **empleado**
    - > **pedido**
    - > **producto**
    - > **restaurante**
  - > **Trigger Functions**
  - > **Types**

Dashboard Properties SQL Statistics Dependencies Dependents Processes postgres/postgres@Parcial2+ X

Query Query History

```

    INSERT INTO empleado (id_empleado, nombre, rol, id_rest) VALUES
    (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),
    (16, 'Manuel Torres', 'Cocinero', 4),
    (17, 'Tatiana Salazar', 'Cajero', 4),
    (18, 'Felipe Gutiérrez', 'Administrador', 4),
    (19, 'Eliana Romero', 'Ayudante', 4),
    (20, 'Esteban Cárdenas', 'Domiciliario', 4),
  
```

Data Output Messages Notifications

INSERT 0 58

Query returned successfully in 163 msec.

Total rows: Query complete 00:00:00.163 ✓ Query returned successfully in 163 msec. X CRLF Ln 1, Col 14

https://supabase.com/dashboard/project/uioactaqcnxnamgmi/editor/17256?schema=public

alejanhuert109 Free PARCIALBd2 Connect Enable branching

Table Editor

	id_empleado	nombre	rol	id_rest
1	1	Laura Mendoza	Cocinero	1
2	2	Carlos Ramírez	Cajero	1
3	3	Marta Gómez	Administrador	1
4	4	David Rodríguez	Domiciliario	1
5	5	Sandra Beltrán	Ayudante	1
6	6	Oscar Pérez	Cocinero	2
7	7	Natalia Quintero	Cajero	2
8	8	Jorge Suárez	Administrador	2
9	9	Diana López	Ayudante	2
10	10	Luis Herrera	Domiciliario	2
11	11	Camila Vargas	Cocinero	3
12	12	Andrés Díaz	Cajero	3
13	13	Paola Castaño	Administrador	3
14	14	Santiago Ríos	Domiciliario	3
15	15	Karla Jiménez	Ayudante	3

Page 1 of 1 100 rows 50 records Refresh Data Definition

## Tabla producto

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

- graphql\_public
- pgbouncer
- public
  - Aggregates
  - Collations
  - FTS Configurations
  - FTS Dictionaries
  - FTS Parsers
  - FTS Templates
  - Foreign Tables
  - Functions
  - Materialized Views
  - Operators
  - Procedures
  - Sequences
  - Tables (5)
    - detalleepedido
    - empleado
    - pedido
    - producto
    - restaurante
  - Trigger Functions
  - Types
  - Views
- realtime
- storage
- vault
- Subscriptions

Login/Group Roles

Dashboard X Properties X SQL X Statistics X Dependencies X Dependents X Processes X postgres/postgres@Parcial2+ X

Query History

```

1 ✓ INSERT INTO Producto (id_prod, nombre, precio) VALUES
2 (1, 'Hamburguesa Clásica', 15000),
3 (2, 'Hamburguesa BBQ', 17000),
4 (3, 'Hamburguesa Doble Carne', 20000),
5 (4, 'Papas Fritas Pequeñas', 6000),
6 (5, 'Papas Fritas Grandes', 9000),
7 (6, 'Gaseosa 350ml', 4000),
8 (7, 'Gaseosa 600ml', 5500),
9 (8, 'Aqua sin gas', 3000),
10 (9, 'Jugo natural', 5000),
11 (10, 'Taco de Pollo', 7000),
12 (11, 'Taco de Carne', 7500),
13 (12, 'Taco Mixto', 8000),
14 (13, 'Burrito Clásico', 12000),
15 (14, 'Burrito Vegetariano', 11000),
16 (15, 'Nachos con Queso', 8500),
17 (16, 'Alitas BBQ (6 unidades)', 10000),
18 (17, 'Alitas Picantes (6 unidades)', 10000),
19 (18, 'Combo Hamburguesa + Papas + Gaseosa', 22000),
20 (19, 'Combo Taco + Nachos + Jugo', 18000),
21 (20, 'Helado de Vainilla', 4500),

```

Data Output Messages Notifications

INSERT 0 50

Query returned successfully in 113 msec.

Total rows: Query complete 00:00:00.113

✓ Query returned successfully in 113 msec. CRLF - Ln 52, Col 1

https://supabase.com/dashboard/project/uioactaqyvnvxnamgmi/editor/17269?schema=public

alejanhuer109 Free PARCIALbd2 Connect Enable branching Feedback RLS disabled Rule postgres Realtime off API Docs

Table Editor

	id...	nombre	precio
	1	Hamburguesa Clásica	15000.00
	2	Hamburguesa BBQ	17000.00
	3	Hamburguesa Doble Carne	20000.00
	4	Papas Fritas Pequeñas	6000.00
	5	Papas Fritas Grandes	9000.00
	6	Gaseosa 350ml	4000.00
	7	Gaseosa 600ml	5500.00
	8	Aqua sin gas	3000.00
	9	Jugo natural	5000.00
	10	Taco de Pollo	7000.00
	11	Taco de Carne	7500.00
	12	Taco Mixto	8000.00
	13	Burrito Clásico	12000.00
	14	Burrito Vegetariano	11000.00
	15	Nachos con Queso	8500.00

Page 1 of 1 100 rows 100 records Refresh Data Definition

## Tabla pedido

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

- > **yuioactacycnvxnamgmi**
  - > **graphql\_public**
  - > **pgbouncer**
  - > **public**
    - > **Aggregates**
    - > **Collations**
    - > **Domains**
    - > **FTS Configurations**
    - > **FTS Dictionaries**
    - > **FTS Parsers**
    - > **FTS Templates**
    - > **Foreign Tables**
    - > **Functions**
    - > **Materialized Views**
    - > **Operators**
    - > **Procedures**
    - > **Sequences**
    - > **Tables (5)**
      - > **detaliepedido**
      - > **empleado**
      - > **pedido**
        - > **Columns (4)**
          - id\_pedido**
          - fecha**
          - id\_rest**
          - total**
        - > **Constraints**
        - > **Indexes**
        - > **RLS Policies**
        - > **Rules**
        - > **Triggers**

Dashboard X Properties X SQL X Statistics X Dependencies X Dependents X Processes X **postgres/postgres@Parcial2\*** X

Query Query History

```

1 ✓ INSERT INTO pedido (id_pedido, fecha, id_rest, total) VALUES
2 (1, '2025-04-01', 1, 150.00),
3 (2, '2025-04-02', 1, 200.00),
4 (3, '2025-04-03', 2, 120.00),
5 (4, '2025-04-04', 2, 320.00),
6 (5, '2025-04-05', 3, 250.00),
7 (6, '2025-04-06', 3, 180.00),
8 (7, '2025-04-07', 1, 160.00),
9 (8, '2025-04-08', 2, 135.00),
10 (9, '2025-04-09', 3, 220.00),
11 (10, '2025-04-10', 1, 190.00),
12 (11, '2025-04-11', 1, 175.00),
13 (12, '2025-04-12', 2, 95.00),
14 (13, '2025-04-13', 3, 200.00),
15 (14, '2025-04-14', 1, 145.00),
16 (15, '2025-04-15', 2, 210.00),
17 (16, '2025-04-16', 3, 300.00),
18 (17, '2025-04-17', 1, 80.00),
19 (18, '2025-04-18', 2, 400.00),
20 (19, '2025-04-19', 1, 150.00),
21 (20, '2025-04-20', 3, 180.00),

```

Data Output Messages Notifications

INSERT 0 50

Query returned successfully in 120 msec.

Total rows: 21 Query complete 00:00:00.120 ✓ Query returned successfully in 120 msec. X CRLF Ln 51, Col 33

https://supabase.com/dashboard/project/yuioactacycnvxnamgmi/editor/17276?schema=public

alejanhuert109 Free PARCIALbd2 Connect Enable branching Todos los favoritos

Table Editor

schema public

+ New table

Search tables...

detaliepedido empleado pedido producto restaurante

Insert

	<b>id_pedido</b>	<b>fecha</b>	<b>id_rest</b>	<b>total</b>
1	2025-04-01	1	150.00	
2	2025-04-02	1	200.00	
3	2025-04-03	2	120.00	
4	2025-04-04	2	320.00	
5	2025-04-05	3	250.00	
6	2025-04-06	3	180.00	
7	2025-04-07	1	160.00	
8	2025-04-08	2	135.00	
9	2025-04-09	3	220.00	
10	2025-04-10	1	190.00	
11	2025-04-11	1	175.00	
12	2025-04-12	2	95.00	
13	2025-04-13	3	200.00	
14	2025-04-14	1	145.00	
15	2025-04-15	2	210.00	

Page 1 of 1 100 rows 50 records Refresh Data Definition

## Tabla DetallePedido

```

1 v INSERT INTO DetallePedido (id_detalle, id_pedido, id_prod, cantidad, subtotal) VALUES
2 (1, 1, 1, 2, 30.00),
3 (2, 1, 2, 1, 60.00),
4 (3, 2, 3, 1, 100.00),
5 (4, 2, 2, 1, 120.00),
6 (5, 3, 1, 4, 80.00),
7 (6, 3, 3, 2, 240.00),
8 (7, 4, 2, 1, 320.00),
9 (8, 5, 1, 5, 150.00),
10 (9, 5, 3, 2, 250.00),
11 (10, 6, 1, 3, 90.00),
12 (11, 7, 2, 2, 120.00),
13 (12, 7, 3, 1, 180.00),
14 (13, 8, 1, 1, 80.00),
15 (14, 9, 2, 1, 135.00),
16 (15, 9, 3, 1, 220.00),
17 (16, 10, 1, 2, 60.00),
18 (17, 11, 2, 2, 350.00),
19 (18, 12, 1, 1, 90.00),
20 (19, 13, 3, 3, 180.00),
21 (20, 14, 2, 1, 175.00),
22 (21, 15, 1, 4, 120.00),
23 (22, 16, 2, 1, 300.00),
24 (23, 17, 1, 3, 240.00),
25 (24, 18, 2, 2, 280.00),
26 (25, 19, 3, 1, 100.00),
27 (26, 20, 1, 2, 200.00),
28 (27, 21, 2, 1, 230.00),
29 (28, 22, 3, 2, 340.00),

```

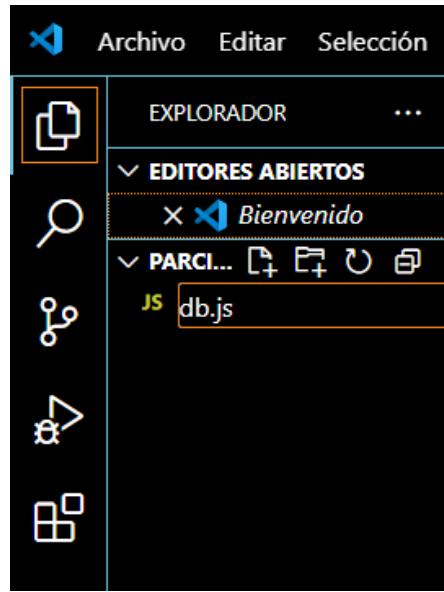
	id_detalle	id_pedido	id_prod	cantidad	subtotal
1	1	1	1	2	30.00
2	1	1	2	1	60.00
3	2	2	3	1	100.00
4	2	2	2	1	120.00
5	3	3	1	4	80.00
6	3	3	2	2	240.00
7	4	2	1	1	320.00
8	5	1	5	1	150.00
9	5	3	2	2	250.00
10	6	1	3	1	90.00
11	7	2	2	2	120.00
12	7	3	1	1	180.00
13	8	1	1	1	80.00
14	9	2	1	1	135.00
15	9	3	1	1	220.00
16	10	1	2	1	60.00
17	11	2	2	2	350.00
18	12	1	1	1	90.00
19	13	3	3	3	180.00
20	14	2	1	1	175.00
21	15	1	4	2	120.00
22	16	2	1	1	300.00
23	17	1	3	2	240.00
24	18	2	2	2	280.00
25	19	3	1	1	100.00
26	20	1	2	2	200.00
27	21	2	1	1	230.00
28	22	3	2	2	340.00

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**.

```
// Importamos el cliente de PostgreSQL
const { Pool } = require('pg');

// Configuración de la conexión a Supabase
const pool = new Pool({
  host: 'aws-0-us-east-1.pooler.supabase.com',
  port: 5432,
  user: 'postgres.ulioactaqycnvxnamgwi',
  password: 'Pa7a16050',
  database: 'postgres',
  ssl: { rejectUnauthorized: false } // Supabase requiere SSL
});

// Probar la conexión
pool.connect((err, client, release) => {
  if (err) {
    console.error('Error conectando con la base de datos', err.stack);
  } else {
    console.log('Conectado a la base de datos');
    release(); // Libera el cliente después de usarlo
  }
});

module.exports=pool;
```

**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.

A screenshot of the VS Code terminal tab. The command 'npm init -y' is being run in a Windows command prompt (PS). The output shows the creation of a package.json file with default values.

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.

A screenshot of the VS Code terminal tab. The command 'npm i express pg dotenv cors' is being run in a Windows command prompt (PS). The output shows the installation of 83 packages, auditing 84 packages in 11 seconds, and finding 0 vulnerabilities.

**17.** Se utiliza **node --watch index.js** en la terminal para que se detecten los cambios 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 X
JS index.js > ...
1 //Jalo la libreria express
2 const express = require('express');
3 //jalo la conexión 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 a los json
12 app.use(express.json());
13
```

**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

```
39 //CRUD RESTAURANTE
40
41 //MÉTODO POST
42
43 app.post('/api/guardar', (req, res) => {
44   const { id_rest, nombre, ciudad, direccion, fecha_apertura } = req.body;
45   const query = 'INSERT INTO Restaurante (id_rest, nombre, ciudad, direccion, fecha_apertura) VALUES ($1, $2, $3, $4, $5)';
46
47   connection.query(query, [id_rest, nombre, ciudad, direccion, fecha_apertura], (error, result) => {
48     if (error) {
49       res.status(500).json({
50         message: 'ERROR CREANDO EL RESTAURANTE',
51         error
52       });
53     } else {
54       res.status(201).json({ id_rest, nombre, ciudad, direccion, fecha_apertura });
55     }
56   });
57 });


```

## Get

```
60 //MÉTODO GET
61
62 app.get('/api/obtener', (req, res) => {
63   const query = 'SELECT * FROM Restaurante';
64
65   connection.query(query, (error, result) => {
66     if (error) {
67       res.status(500).json({
68         success: false,
69         message: "Error al recuperar los datos",
70         details: error.message
71       });
72     } else {
73       res.status(200).json({
74         success: true,
75         message: "Datos de la tabla",
76         details: result
77       });
78     }
79   });
80 });


```

## Put

```

82 //MÉTODO PUT
83 app.put('/api/actualizar/:id_rest', (req, res) => {
84   const { id_rest } = req.params;
85   const { nombre, ciudad, direccion, fecha_apertura } = req.body;
86
87   const query = `UPDATE Restaurante SET nombre = $1, ciudad = $2, direccion = $3, fecha_apertura = $4 WHERE id_rest = $5`;
88
89   connection.query(query, [nombre, ciudad, direccion, fecha_apertura, id_rest], (error, result) => {
90     if (error) {
91       res.status(500).json({
92         success: false,
93         message: 'Error al actualizar el registro',
94         details: error.message
95       });
96     } else if (result.rowCount === 0) {
97       res.status(404).json({
98         success: false,
99         message: 'No se encontró ningún registro con el id ${id_rest}'
100      });
101    } else {
102      res.status(200).json({
103        success: true,
104        message: 'Registro actualizado correctamente',
105        updated: {
106          id_rest,
107          nombre,
108          ciudad,
109          direccion,
110          fecha_apertura
111        }
112      });
113    }
114  });

```

## Delete

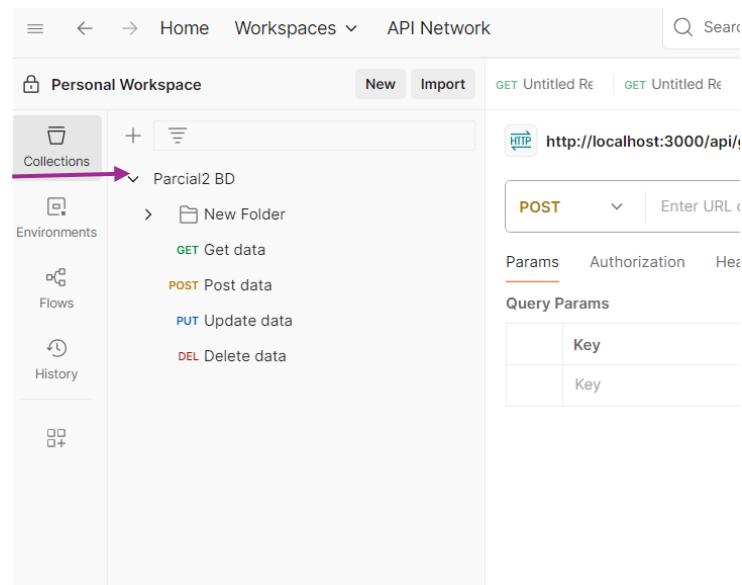
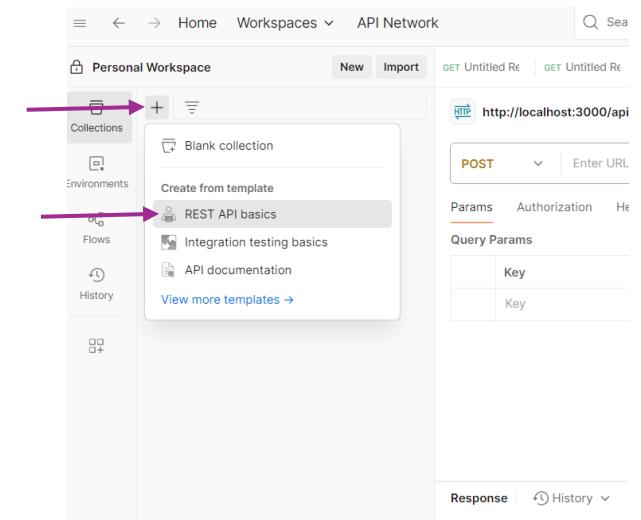
```

118 //MÉTODO DELETE
119
120 app.delete('/api/eliminar/:id_rest', (req, res) => {
121   const { id_rest } = req.params;
122   const query = 'DELETE FROM Restaurante WHERE id_rest = $1';
123
124   connection.query(query, [id_rest], (error, result) => {
125     if (error) {
126       res.status(500).json({
127         success: false,
128         message: 'Error al eliminar el registro',
129         details: error.message
130       });
131     } else if (result.rowCount === 0) {
132       res.status(404).json({
133         success: false,
134         message: 'No existe el registro ${id_rest}',
135       });
136     } else {
137       res.status(200).json({
138         success: true,
139         message: 'Dato eliminado de la tabla',
140         details: result
141       });
142     }
143   });
144 });
145 });
146 });
147 });

```

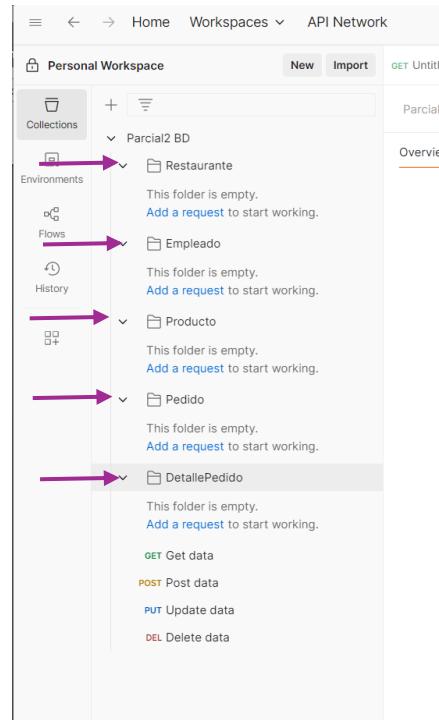
24. En la interfaz de Postman se crea una nueva colección, en este caso

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

```

POST http://localhost:3000/api/guardar
{
  "id_rest": "51",
  "nombre": "La Casa Burger",
  "direccion": "Calle 13",
  "ciudad": "Cali",
  "telefono": "3124567898",
  "tipoComida": "Mexicana",
  "fecha_apertura": "2023-01-01"
}
    
```

#### Get

Personal Workspace

Parcial2 BD / Restaurante / GET

GET http://localhost:3000/api/obtener

```

1 {
  "success": true,
  "message": "Registros de la tabla",
  "detail": {
    "command": "SELECT",
    "rowCount": 51,
    "oid": null,
    "rows": [
      {
        "id_rest": 1,
        "nombre": "La Cabaña del Sabor",
        "ciudad": "Bogotá"
      }
    ]
  }
}

```

200 OK · 1.67 s · 8.67 KB

## Put

Personal Workspace

Parcial2 BD / PUT

PUT http://localhost:3000/api/actualizar/45

```

1 {
  "id_rest": 45,
  "nombre": "Build Dogs",
  "direccion": "Centro",
  "ciudad": "Pacho",
  "telefono": "3156279345",
  "tipoComida": "Comidas rápidas",
  "fecha_apertura": "2025-01-12"
}

```

200 OK · 1.32 s · 414 B

## Delete

The screenshot shows the Postman interface. On the left, the 'Personal Workspace' sidebar lists collections: 'Parcial2 BD' (with 'Restaurante' and 'Empleado' sub-folders), 'Flows', 'History', and 'Pedido'. The main area displays a 'DELETE' request to 'http://localhost:3000/api/eliminar/51'. The 'Body' tab shows a JSON response with the following content:

```

1   {
2     "success": true,
3     "message": "Dato eliminado de la tabla",
4     "details": [
5       {
6         "command": "DELETE",
7         "rowCount": 1,
8         "id": null,
9         "rows": [],
10        "fields": [],
11        "_types": {
12          "_types": {
13            "arrayParser": false
14          }
15        }
16      }
17    ]
18  }

```

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

## Post

The screenshot shows the Supabase Table Editor for the 'restaurante' table. The table has columns: id\_restaurante, nombre, ciudad, and direccion. The rows are numbered 37 to 51. A red arrow points to the last row (id 51), which contains the values: 'La Casa Burguer', 'Cali', 'Calle 13', and 'Calle 13'. The table editor interface includes a sidebar with tables like detallepedido, empleado, pedido, producto, and restaurante.

	id_restaurante	nombre	ciudad	direccion
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	
51	La Casa Burguer	Cali	Calle 13	

## Put

Table Editor

schema public

+ New table

Search tables...

**restaurante**

Insert

RLS disabled Role postgres Realtime off API

	id_r...	nombre	ciudad	direccion
	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

Table Editor

schema public

+ New table

Search tables...

**restaurante**

Insert

RLS disabled Role postgres Realtime off API

	id_r...	nombre	ciudad	direccion	fecha_apertura
	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

	id_restaurante	nombre	ciudad	direccion	fecha_apertura
	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

	id_restaurante	nombre	ciudad	direccion	fecha_apertura
	36	La Rápida	Pasto	Cra 6 #10-15	2022-06-20
	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

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

## Post

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

## Get

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

## Put

```

194 //MÉTODO PUT
195
196 app.put('/api/actualizar_empleado/:id_empleado', (req, res) => {
197   const { id_empleado } = req.params;
198   const { nombre, rol, id_rest } = req.body;
199
200   const query = 'UPDATE empleado SET nombre = $1, rol = $2, id_rest = $3 WHERE id_empleado = $4';
201
202   connection.query(query, [nombre, rol, id_rest, id_empleado], (error, result) => {
203     if (error) {
204       res.status(500).json({
205         success: false,
206         message: 'Error al actualizar el registro',
207         details: error.message
208       });
209     } else if (result.rowCount === 0) {
210       res.status(404).json({
211         success: false,
212         message: `No se encontró ningún empleado con el id ${id_empleado}`
213       });
214     } else {
215       res.status(200).json({
216         success: true,
217         message: 'Empleado actualizado correctamente',
218         updated: {
219           id_empleado,
220           nombre,
221           rol,
222           id_rest
223         }
224       });
225     }
226   });

```

## Delete

```

231 //MÉTODO DELETE
232
233 app.delete('/api/eliminar_empleado/:id_empleado', (req, res) => {
234   const { id_empleado } = req.params;
235   const query = 'DELETE FROM Empleado WHERE id_empleado = $1';
236
237   connection.query(query, [id_empleado], (error, result) => {
238
239     if (error) {
240       res.status(500).json({
241         success: false,
242         message: "Error al eliminar el registro",
243         details: error.message
244       });
245     } else if (result.rowCount === 0) {
246       res.status(404).json({
247         success: false,
248         message: `No existe el registro ${id_empleado}`,
249       });
250     } else {
251       res.status(200).json({
252         success: true,
253         message: "Dato eliminado de la tabla",
254         details: result
255       });
256     }
257   });
258 });
259 });
260

```

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

## Post

The screenshot shows the Postman interface with a collection named "Parcial2 BD". A POST request is selected for the "Empleado" endpoint. The request URL is `http://localhost:3000/api/guardar_empleado`. The "Body" tab is active, showing the following JSON payload:

```
1 {  
2     "id_empleado": 52,  
3     "nombre": "Juan Carlos Molina",  
4     "rol": "Administrador",  
5     "id_rest": 24  
6 }  
7
```

The response status is `201 Created` with a response time of 1.08 s and a size of 323 B. The response body is identical to the sent body.

## Get

The screenshot shows the Postman interface with a collection named "Parcial2 BD". A GET request is selected for the "Empleado" endpoint. The request URL is `http://localhost:3000/api/obtener_empleado`. The "Body" tab is active, showing an empty JSON object: `{}`.

The response status is `200 OK` with a response time of 1.13 s and a size of 5.64 KB. The response body contains JSON data about an employee:

```
2     "success": true,  
3     "message": "Datos de la tabla",  
4     "details": {  
5         "command": "SELECT",  
6         "rowCount": 51,  
7         "oid": null,  
8         "rows": [  
9             {  
10                 "id_empleado": 1,  
11                 "nombre": "Laura Mendoza",  
12                 "rol": "Cocinero",  
13                 "id_rest": 1  
14             }  
15         ]  
16     }  
17 }
```

## Put

The screenshot shows the Postman interface with a collection named "Parcial2 BD". A PUT request is being made to the URL `http://localhost:3000/api/actualizar_empleado/5`. The request body contains the following JSON:

```

1 {
2   "id_empleado": 5,
3   "nombre": "Javier Ospina",
4   "rol": "Limpieza general",
5   "id_rest": 1
6 }

```

The response status is 200 OK, indicating success. The response body is:

```

1 {
2   "success": true,
3   "message": "Empleado actualizado correctamente",
4   "updated": {
5     "id_empleado": 5,
6     "nombre": "Javier Ospina",
7     "rol": "Limpieza general",
8     "id_rest": 1
9   }
10 }

```

## Delete

The screenshot shows the Postman interface with a collection named "Parcial2 BD". A DELETE request is being made to the URL `http://localhost:3000/api/eliminar_empleado/47`. The request body is empty. The response status is 200 OK, indicating success. The response body is:

```

1 {
2   "success": true,
3   "message": "Dato eliminado de la tabla",
4   "details": {
5     "command": "DELETE",
6     "rowCount": 1,
7     "oid": null,
8     "rows": [],
9     "fields": [],
10     "_types": {
11       "_types": {
12         "arrayParser": null
13       }
14     }
15   }
16 }

```

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

## Post

Table Editor

Filter Sort Insert

schema public

New table

Search tables...

detallepedido

empleado

pedido

producto

restaurante

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 Calcedo Cocinero 9

42 Camilo Sánchez Cajero 9

43 Lorena Torres Administrador 9

44 Juan Valderrama Ayudante 9

45 Nately 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 Define

## Put

Table Editor

Filter Sort Insert

schema public

New table

Search tables...

detallepedido

empleado

pedido

producto

restaurante

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

Table Editor

Filter Sort Insert

schema public

New table

Search tables...

detallepedido

empleado

pedido

producto

restaurante

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

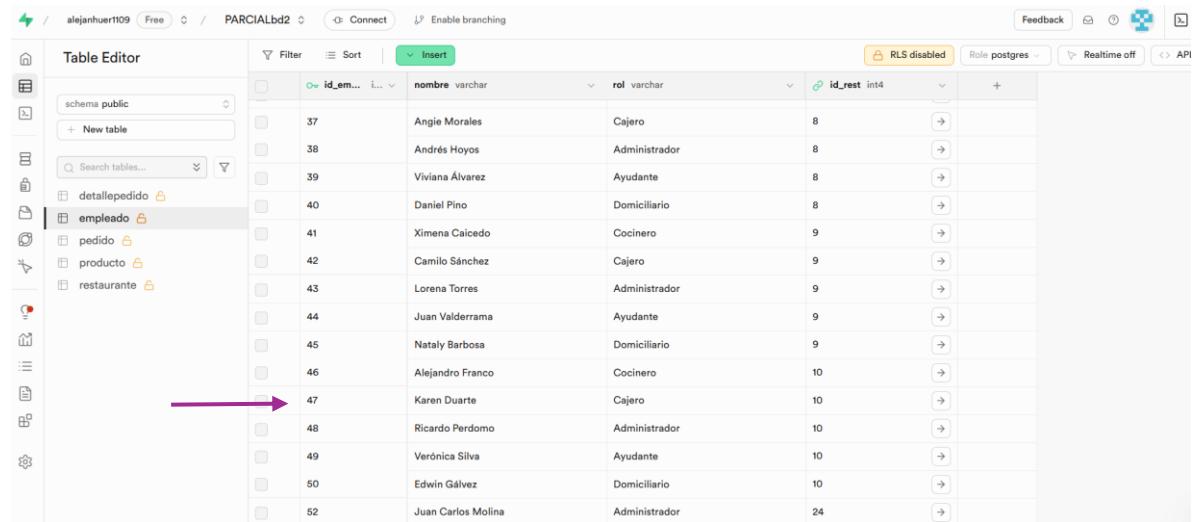
12 Andrés Díaz Cajero 3

13 Paola Castaño Administrador 3

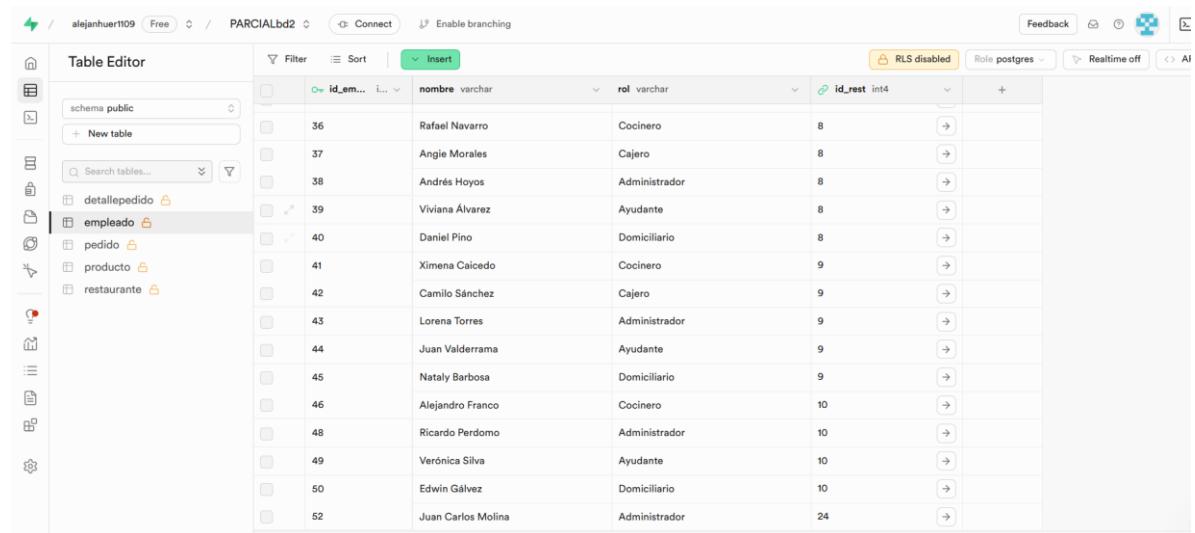
14 Santiago Ríos Domiciliario 3

15 Karla Jiménez Ayudante 3

## Delete



	id_empleado	nombre	rol	id_rest	
	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	



	id_empleado	nombre	rol	id_rest	
	36	Rafael Navarro	Cocinero	8	
	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	
	48	Ricardo Perdomo	Administrador	10	
	49	Verónica Silva	Ayudante	10	
	50	Edwin Gálvez	Domiciliario	10	
	52	Juan Carlos Molina	Administrador	24	

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

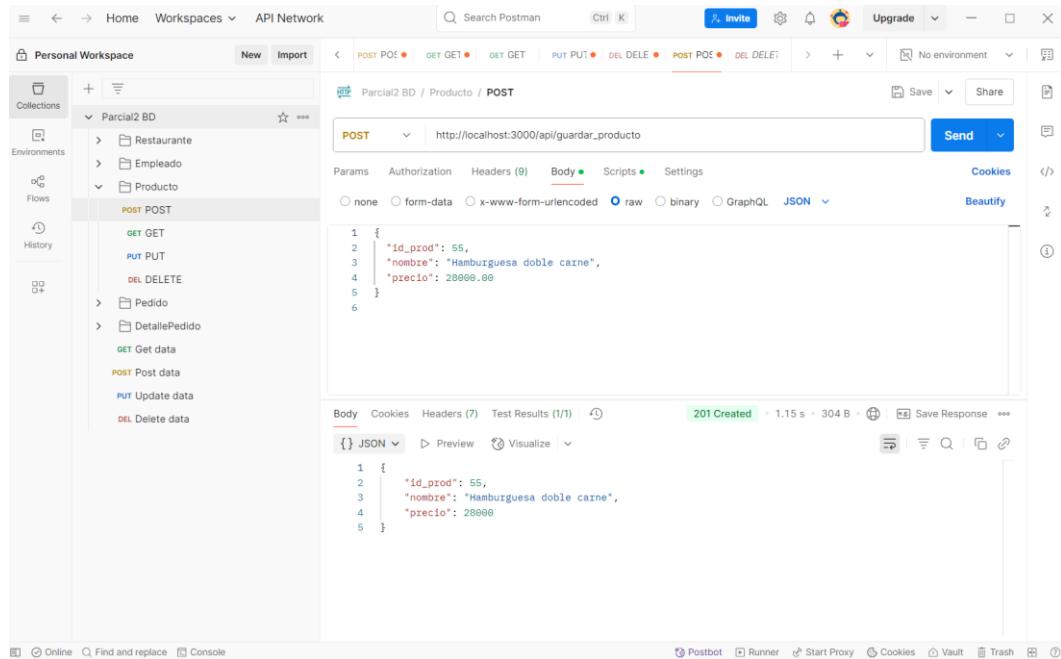
```

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     } else {
362       res.status(200).json({
363         success: true,
364         message: "Producto eliminado de la tabla",
365         details: result
366       });
367     }
368   });
369 });
370 });
371 });

```

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

#### Post

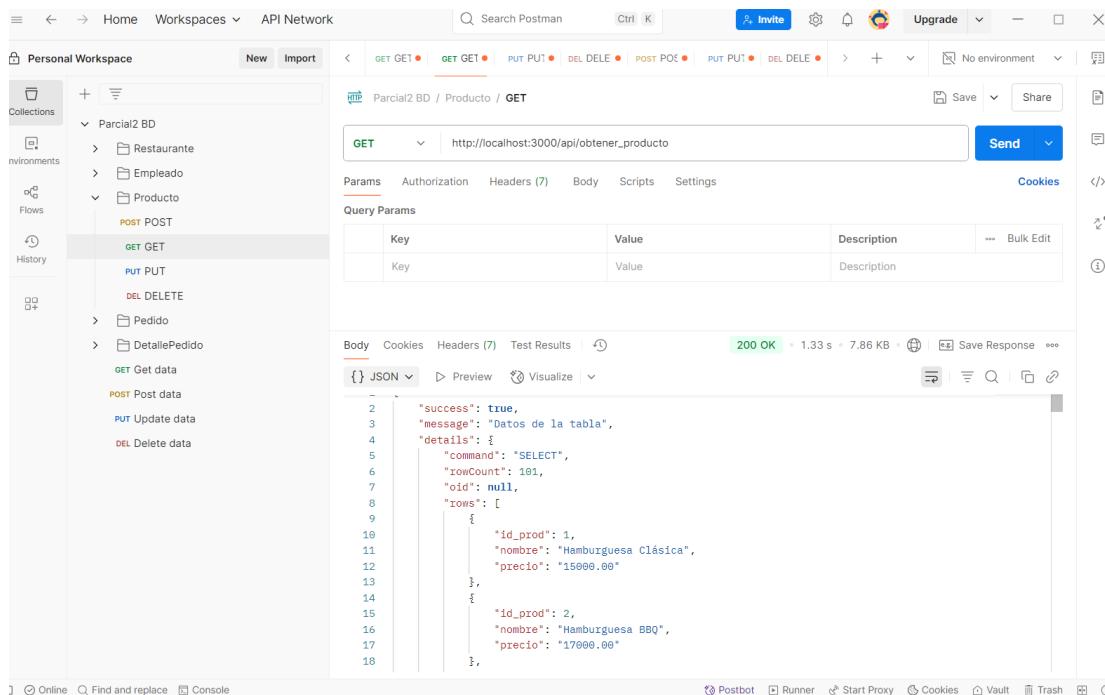


The screenshot shows the Postman interface with a collection named "Parcial2 BD" selected. Under the "Producto" folder, there is a POST request labeled "POST POST". The request URL is `http://localhost:3000/api/guardar_producto`. The "Body" tab is selected, showing the following JSON payload:

```
1 {
2     "id_prod": 55,
3     "nombre": "Hamburguesa doble carne",
4     "precio": 28000.00
5 }
```

The response status is `201 Created`, and the response body is identical to the sent body.

#### Get



The screenshot shows the Postman interface with a collection named "Parcial2 BD" selected. Under the "Producto" folder, there is a GET request labeled "GET GET". The request URL is `http://localhost:3000/api/obtener_producto`. The "Headers" tab is selected, showing the following query parameters:

Key	Value	Description
Key	Value	Description

The response status is `200 OK`, and the response body is a JSON object containing success status, message, details, and rows:

```
1 {
2     "success": true,
3     "message": "Datos de la tabla",
4     "details": {
5         "command": "SELECT",
6         "rowCount": 101,
7         "oid": null,
8         "rows": [
9             {
10                 "id_prod": 1,
11                 "nombre": "Hamburguesa Clásica",
12                 "precio": "15000.00"
13             },
14             {
15                 "id_prod": 2,
16                 "nombre": "Hamburguesa BBQ",
17                 "precio": "17000.00"
18             }
19         ]
20     }
21 }
```

## Put

The screenshot shows the Postman interface with a PUT request to `http://localhost:3000/api/actualizar_producto/121`. The request body contains the following JSON:

```
1 {
2   "id_prod": 121,
3   "nombre": "Helado de oreo",
4   "precio": 2000.00
5 }
```

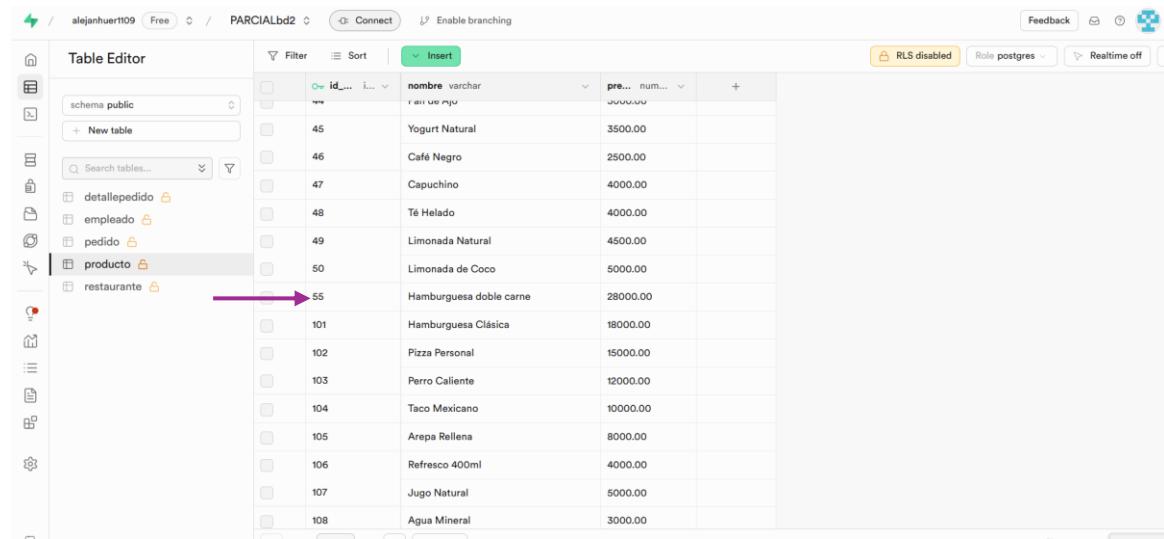
The response status is 200 OK, with the message: "success": true, "message": "Producto actualizado correctamente", "updated": {"id\_prod": "121", "nombre": "Helado de oreo", "precio": 2000}

## Delete

The screenshot shows the Postman interface with a DELETE request to `http://localhost:3000/api/eliminar_producto/26`. The response status is 200 OK, with the message: "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}}]}]}

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

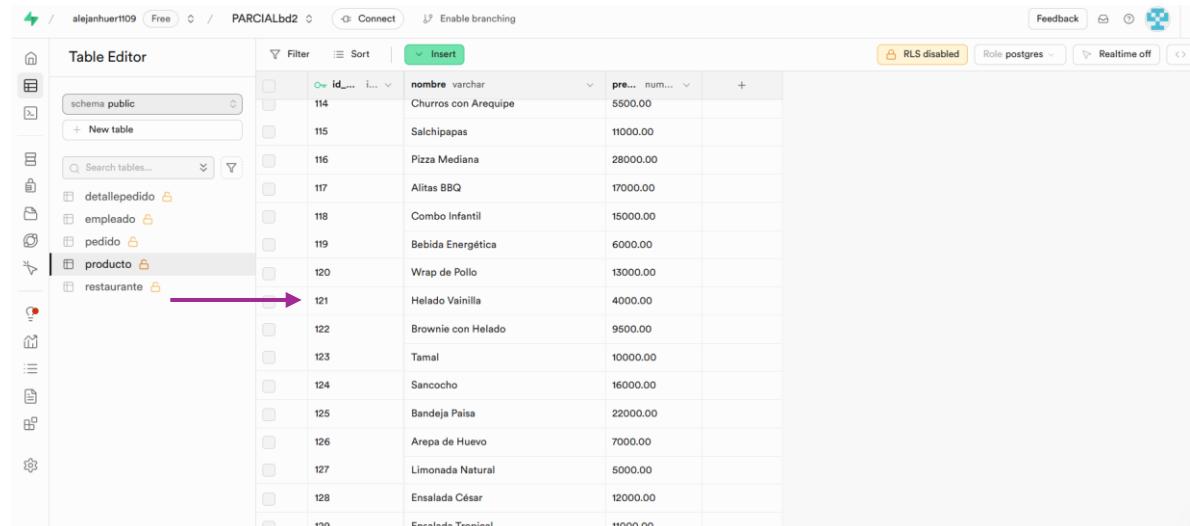
## Post



The screenshot shows the Supabase Table Editor interface for the 'producto' table. The table has columns: id (number), nombre (varchar), and precio (number). The data includes various food items with their names and prices. A purple arrow points to the last row, which was added via a POST request.

	id	nombre	precio
	1	Leche UHT	3000.00
	45	Yogurt Natural	3500.00
	46	Café Negro	2500.00
	47	Capuchino	4000.00
	48	Té Helado	4000.00
	49	Limonada Natural	4500.00
	50	Limonada de Coco	5000.00
	55	Hamburguesa doble carne	28000.00
	101	Hamburguesa Clásica	18000.00
	102	Pizza Personal	15000.00
	103	Perro Caliente	12000.00
	104	Taco Mexicano	10000.00
	105	Arepa Rellena	8000.00
	106	Refresco 400ml	4000.00
	107	Jugo Natural	5000.00
	108	Agua Mineral	3000.00

## Put



The screenshot shows the Supabase Table Editor interface for the 'producto' table. The table has columns: id (number), nombre (varchar), and precio (number). The data includes various food items with their names and prices. A purple arrow points to the last row, which was updated via a PUT request.

	id	nombre	precio
	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 Vainilla	4000.00
	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
	129	Ensalada Tropical	11000.00

Table Editor

Search tables...

schema public

+ New table

detallepedido

empleado

pedido

**producto**

restaurante

Filter Sort

	id...	nombre	pre...	num...
	114	Churros con Arequipa	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	Bandejita Paisa	22000.00	
	126	Arepas de Huevo	7000.00	
	127	Limonada Natural	5000.00	
	128	Ensalada César	12000.00	

## Delete

Table Editor

Search tables...

schema public

+ New table

detallepedido

empleado

pedido

**producto**

restaurante

Filter Sort

	id...	nombre	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	Arepas Rellenas	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

The screenshot shows a PostgreSQL Table Editor interface. On the left, there's a sidebar with various icons for database management tasks like creating tables, managing schemas, and viewing logs. The main area is titled 'Table Editor' and shows the 'producto' table. The table has columns: id\_producto (integer), nombre (varchar), and precio (numerical). The data consists of 31 rows, each representing a different food item with its name and price.

	id_producto	nombre	precio
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	
27	Chorizo con Arepa	7500.00	
28	Chicharrón con Yuca	10000.00	
29	Tamales	9500.00	
30	Huevos Pericos con Pan	7000.00	
31	Calentado de Frijoles	9000.00	

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

## Post

```

373 //CRUD PEDIDO
374
375 //MÉTODO POST
376
377 app.post('/api/guardar_pedido', (req, res) => {
378   const { id_pedido, fecha, id_rest, total } = req.body;
379   const query = 'INSERT INTO Pedido (id_pedido, fecha, id_rest, total) VALUES ($1, $2, $3, $4)';
380
381   connection.query(query, [id_pedido, fecha, id_rest, total], (error, result) => {
382     if (error) {
383       res.status(500).json({
384         message: 'ERROR CREANDO EL PEDIDO',
385         error
386       });
387     } else {
388       res.status(201).json({ id_pedido, fecha, id_rest, total });
389     }
390   });
391 });
392

```

## 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 //MÉTODO 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     } else {
472       res.status(200).json([
473         {
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

POST /api/guardar\_pedido

```
{
  "id_pedido": 51,
  "fecha": "2025-04-10",
  "id_rest": 9,
  "total": 280000.00
}
```

201 Created

```
{
  "id_pedido": 51,
  "fecha": "2025-04-10",
  "id_rest": 9,
  "total": 280000
}
```

## Get

GET /api/obtener\_pedido

200 OK

```
{
  "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"
      }
    ]
  }
}
```

## Put

The screenshot shows the Postman interface with a collection named "Parcial2 BD". A PUT request is selected for the "Pedido" endpoint at `http://localhost:3000/api/actualizar_pedido/30`. The request body contains the following JSON:

```

1  {
2    "id_pedido": 30,
3    "fecha": "2025-04-18",
4    "id_rest": 1,
5    "total": 34000.00
6  }

```

The response status is 200 OK, indicating success. The response body is:

```

1  {
2    "success": true,
3    "message": "Pedido actualizado correctamente",
4    "updated": {
5      "id_pedido": "30",
6      "fecha": "2025-04-18",
7      "id_rest": 1,
8      "total": 34000
9    }
10 }

```

## Delete

The screenshot shows the Postman interface with a collection named "Parcial2 BD". A DELETE request is selected for the "Pedido" endpoint at `http://localhost:3000/api/eliminar_pedido/45`. The request body is empty. The response status is 200 OK, indicating success. The response body is:

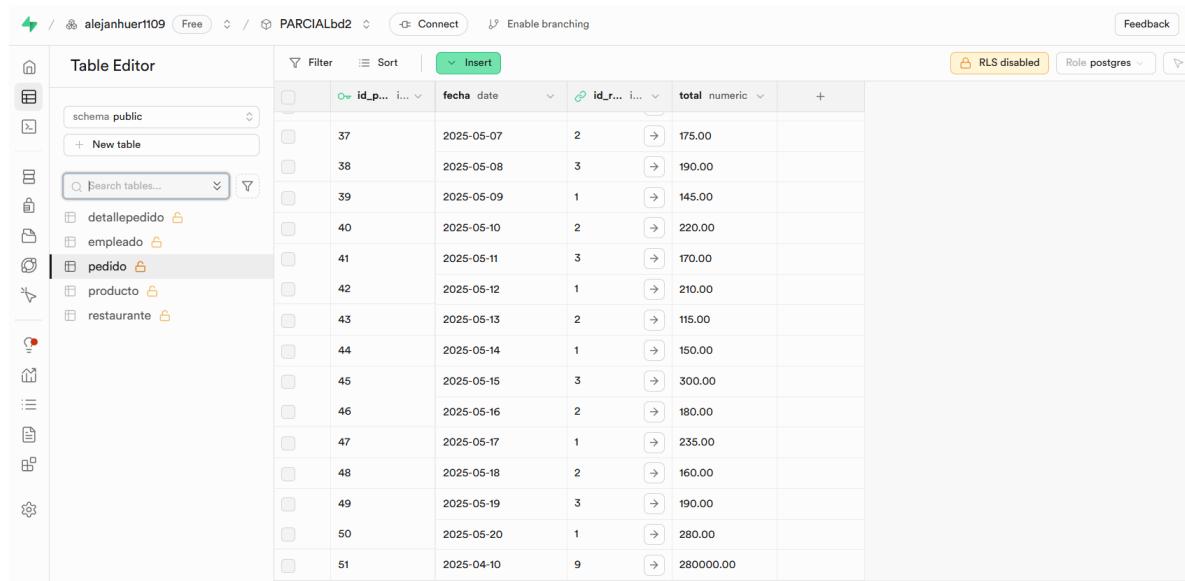
```

1  {
2    "success": true,
3    "message": "Pedido eliminado de la tabla",
4    "details": {
5      "command": "DELETE",
6      "rowCount": 1,
7      "oid": null,
8      "rows": [],
9      "fields": [],
10     "_types": {
11       "-types": {
12         "arrayParser": {},
13         "builtins": {
14           "BOOL": 16,
15           "BYTEA": 17,
16           "CHAR": 18,
17           "INT8": 26,
18           "INT2": 21,
19           "TEXT": 23
20         }
21       }
22     }
23   }

```

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

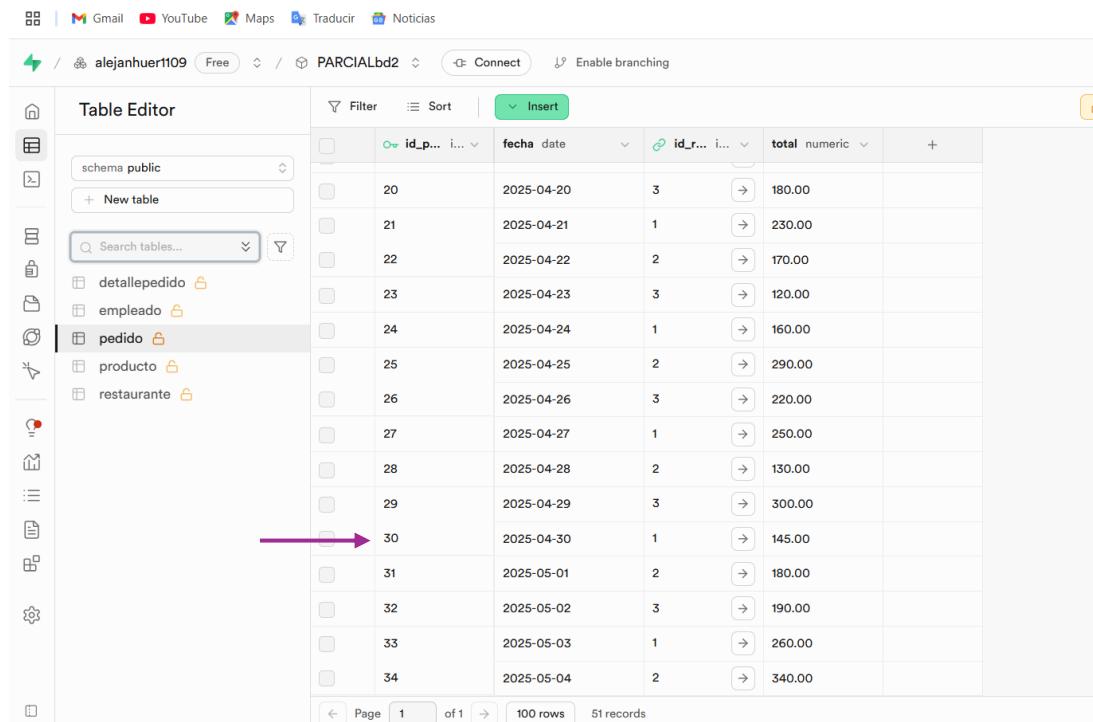
## Post



A screenshot of a PostgreSQL Table Editor interface. The left sidebar shows a tree view of tables: schema public, detallepedido, empleado, pedido (selected), producto, and restaurante. The main area displays a table with columns: id\_pedido, fecha, id\_restaurante, and total. The table contains 51 rows of data, starting from row 37 and ending at row 51. Row 51 has a value of 280000.00 in the 'total' column.

	id_pedido	fecha	id_restaurante	total	
	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



A screenshot of a PostgreSQL Table Editor interface. The left sidebar shows a tree view of tables: schema public, detallepedido, empleado, pedido (selected), producto, and restaurante. The main area displays a table with columns: id\_pedido, fecha, id\_restaurante, and total. The table contains 51 rows of data, starting from row 20 and ending at row 34. A purple arrow points to the 30th record. Row 30 has a value of 145.00 in the 'total' column.

	id_pedido	fecha	id_restaurante	total	
	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	

Table Editor

schema public

+ New table

Search tables...

detallepedido

empleado

**pedido**

producto

restaurante

fecha date

id\_p... i... v

id\_r... i... v

total numeric v

Insert

RLS disabled

Role postgres

51 records

	id_p...	i... v	fecha	date	v	id_r...	i... v	total	numeric	v	+
	20		2025-04-20		v	3		180.00			
	21		2025-04-21		v	1		230.00			
	22		2025-04-22		v	2		170.00			
	23		2025-04-23		v	3		120.00			
	24		2025-04-24		v	1		160.00			
	25		2025-04-25		v	2		290.00			
	26		2025-04-26		v	3		220.00			
	27		2025-04-27		v	1		250.00			
	28		2025-04-28		v	2		130.00			
	29		2025-04-29		v	3		300.00			
	30		2025-04-18		v	1		34000.00			
	31		2025-05-01		v	2		180.00			
	32		2025-05-02		v	3		190.00			
	33		2025-05-03		v	1		260.00			
	34		2025-05-04		v	2		340.00			

## Delete

Table Editor

schema public

+ New table

Search tables...

detallepedido

empleado

**pedido**

producto

restaurante

fecha date

id\_p... i... v

id\_r... i... v

total numeric v

Insert

RLS disabled

51 records

	id_p...	i... v	fecha	date	v	id_r...	i... v	total	numeric	v	+
	37		2025-05-07		v	2		175.00			
	Expand row... 8		2025-05-08		v	3		190.00			
	39		2025-05-09		v	1		145.00			
	40		2025-05-10		v	2		220.00			
	41		2025-05-11		v	3		170.00			
	42		2025-05-12		v	1		210.00			
	43		2025-05-13		v	2		115.00			
	44		2025-05-14		v	1		150.00			
	45		2025-05-15		v	3		300.00			
	46		2025-05-16		v	2		180.00			
	47		2025-05-17		v	1		235.00			
	48		2025-05-18		v	2		160.00			
	49		2025-05-19		v	3		190.00			
	50		2025-05-20		v	1		280.00			
	51		2025-04-10		v	9		280000.00			

	id_detalle	fecha	id_pedido	total
	36	2025-05-06	1	320.00
	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
	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

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

index.js.

## Post

```

483 //CRUD DETALLE PEDIDO
484 //MÉTODO POST
485
486 app.post('/api/guardar_DetallePedido', (req, res) => {
487   const { id_detalle, id_pedido, id_prod, cantidad, subtotal } = req.body;
488   const query = 'INSERT INTO DetallePedido (id_detalle, id_pedido, id_prod, cantidad, subtotal) VALUES ($1, $2, $3, $4, $5)';
489
490   connection.query(query, [id_detalle, id_pedido, id_prod, cantidad, subtotal], (error, result) => {
491     if (error) {
492       res.status(500).json({
493         message: 'ERROR CREANDO EL DETALLE DEL PEDIDO',
494         error
495       });
496     } else {
497       res.status(201).json({
498         success: true,
499         message: 'Detalle del pedido creado correctamente',
500         created: { id_detalle, id_pedido, id_prod, cantidad, subtotal }
501       });
502     }
503   });
504 });
505

```

## Get

```

508
509 //MÉTODO GET
510
511 app.get('/api/obtener_DetallePedido', (req, res) => {
512   const query = 'SELECT * FROM DetallePedido';
513
514   connection.query(query, (error, result) => {
515     if (error) {
516       res.status(500).json({
517         success: false,
518         message: "Error al recuperar los datos de el detella del pedido",
519         details: error.message
520       });
521     } else {
522       res.status(200).json({
523         success: true,
524         message: "Detalles de el pedido",
525         details: result
526       });
527     }
528   });
529 });

```

## Put

```

531 //MÉTODO PUT
532 app.put('/api/actualizar_DetallePedido/:id_detalle', (req, res) => {
533   const { id_detalle } = req.params;
534   const { id_pedido, id_prod, cantidad, subtotal } = req.body;
535
536   const query = 'UPDATE DetallePedido SET id_pedido = $1, id_prod = $2, cantidad = $3, subtotal = $4 WHERE id_detalle = $5';
537
538   connection.query(query, [id_pedido, id_prod, cantidad, subtotal, id_detalle], (error, result) => {
539     if (error) {
540       res.status(500).json({
541         success: false,
542         message: 'Error al actualizar el detalle del pedido',
543         details: error.message
544       });
545     } else if (result.rowCount === 0) {
546       res.status(404).json({
547         success: false,
548         message: `No se encontró ningún detalle de pedido con el id ${id_detalle}`
549       });
550     } else {
551       res.status(200).json({
552         success: true,
553         message: 'Detalle de pedido actualizado correctamente',
554         updated: {
555           id_detalle,
556           id_pedido,
557           id_prod,
558           cantidad,
559           subtotal
560         }
561       });
562     }
563   });

```

PROBLEMAS SAUDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

## Delete

```

567
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     } else {
588       res.status(200).json({
589         success: true,
590         message: "Detalle del pedido eliminado de la tabla",
591         details: result
592       });
593     }
594   });
595 });
596 });
597 });

```

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

#### Post

The screenshot shows the Postman application interface. On the left, the 'Personal Workspace' sidebar lists collections: 'Parcial2 BD' (which contains 'Restaurante', 'Empleado', 'Producto', 'Pedido', and 'DetallePedido'), 'GET GET', 'PUT PUT', 'DEL DELETE', 'GET Get data', 'POST Post data', 'PUT Update data', and 'DEL Delete data'. The 'DetallePedido' collection is currently selected. In the main workspace, a POST request is being prepared for the URL `http://localhost:3000/api/guardar_DetallePedido`. The 'Body' tab is active, showing the following JSON payload:

```

1 {
2   "id_detalle": 51,
3   "id_pedido": 49,
4   "id_prod": 1,
5   "cantidad": 2,
6   "subtotal": 80000.00
7 }

```

Below the body, the response is shown in JSON format:

```

1 {
2   "success": true,
3   "message": "Detalle del pedido creado correctamente",
4   "created": [
5     {
6       "id_detalle": 51,
7       "id_pedido": 49,
8       "id_prod": 1,
9       "cantidad": 2,
10      "subtotal": 80000
11    }
12  ]
13 }

```

## Get

The screenshot shows the Postman interface with a successful GET request. The URL is `http://localhost:3000/api/obtener_DetallePedido`. The response body is a JSON object:

```
1 {  
2   "success": true,  
3   "message": "detalles de el pedido",  
4   "details": [  
5     {"command": "SELECT",  
6     "rowCount": 51,  
7     "oid": null,  
8     "rows": [  
9       {"id_detalle": 1,  
10      "id_pedido": 1,  
11      "id_prod": 1,  
12      "cantidad": 2,  
13      "subtotal": "30.00"  
14    ],  
15    {"id_detalle": 2,  
16      "id_pedido": 1,  
17      "id_prod": 1,  
18      "cantidad": 1,  
19      "subtotal": "30.00"  
20    ]  
21  ]  
22 }  
23 }
```

## Put

The screenshot shows the Postman interface with a successful PUT request. The URL is `http://localhost:3000/api/actualizar_DetallePedido/12`. The request body is a JSON object:

```
1 {  
2   "id_detalle": 12,  
3   "id_pedido": 6,  
4   "id_prod": 1,  
5   "cantidad": 5,  
6   "subtotal": 95500.00  
7 }
```

The response body is a JSON object:

```
1 {  
2   "success": true,  
3   "message": "Detalle de pedido actualizado correctamente",  
4   "updated": [  
5     {"id_detalle": "12",  
6     "id_pedido": 6,  
7     "id_prod": 1,  
8     "cantidad": 5,  
9     "subtotal": 95500  
10   ]  
11 }
```

## Delete

DELETE http://localhost:3000/api/eliminar\_DetallePedido/35

Key	Value	Description
Key	Value	Description

```

1 {
2   "success": true,
3   "message": "Detalle del pedido eliminado de la tabla",
4   "details": {
5     "command": "DELETE",
6     "rowCount": 1,
7     "oid": null,
8     "rows": [],
9     "fields": [],
10    "_types": {
11      "_types": {
12        "arrayParser": {},
13        "builtins": {
14          "BOOL": 16,
15          "BYTEA": 17,
16          "CHAR": 18,
17          "INT8": 28,
18        }
19      }
20    }
21  }
22}
  
```

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

nuestro Supabase.

## Post

	id_detalle	id_pedido	id_prod	cantidad	subtotal
37	31	3	4	380.00	
38	32	1	2	200.00	
39	33	2	1	260.00	
40	34	3	3	210.00	
41	35	1	1	320.00	
42	36	2	2	210.00	
43	37	3	1	170.00	
44	38	1	4	160.00	
45	39	2	2	240.00	
46	40	3	1	190.00	
47	41	1	3	120.00	
48	42	2	1	300.00	
49	43	3	2	150.00	
50	44	1	1	80.00	
51	49	1	2	80000.00	

## Put

Table Editor

Filter Sort Insert

RLS disabled Role postgres Realtime off

	id_detalle	id_pedido	id_prod	cantidad	subtotal
	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

Table Editor

Filter Sort Insert

RLS disabled Role postgres

	id_detalle	id_pedido	id_prod	cantidad	subtotal
	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	6	1	95500.00
	13	8	1	1	80.00
	14	9	2	1	135.00
	15	9	3	1	220.00

## Delete

	<code>id_detalle</code> int4	<code>id_pedido</code> int4	<code>id_prod</code> int4	<code>cantidad</code> int4	<code>subtotal</code> numeric	
	26	20	1	2	200.00	
	27	21	2	1	230.00	
	28	22	3	2	340.00	
	29	23	1	1	120.00	
	30	24	2	2	100.00	
	31	25	3	1	150.00	
	32	26	1	1	180.00	
	33	27	2	3	210.00	
	34	28	3	1	220.00	
	35	29	1	5	400.00	
	36	30	2	1	145.00	
	37	31	3	4	380.00	
	38	32	1	2	200.00	
	39	33	2	1	260.00	
	40	34	3	3	210.00	
	← Page 1 of 1 → 100 rows 51 records					

	<code>id_detalle</code> int4	<code>id_pedido</code> int4	<code>id_prod</code> int4	<code>cantidad</code> int4	<code>subtotal</code> numeric	
	26	20	1	2	200.00	
	27	21	2	1	230.00	
	28	22	3	2	340.00	
	29	23	1	1	120.00	
	30	24	2	2	100.00	
	31	25	3	1	150.00	
	32	26	1	1	180.00	
	33	27	2	3	210.00	
	34	28	3	1	220.00	
	36	30	2	1	145.00	
	37	31	3	4	380.00	
	38	32	1	2	200.00	
	39	33	2	1	260.00	
	40	34	3	3	210.00	
	41	35	1	1	320.00	
	← Page 1 of 1 → 100 rows 50 records					

**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

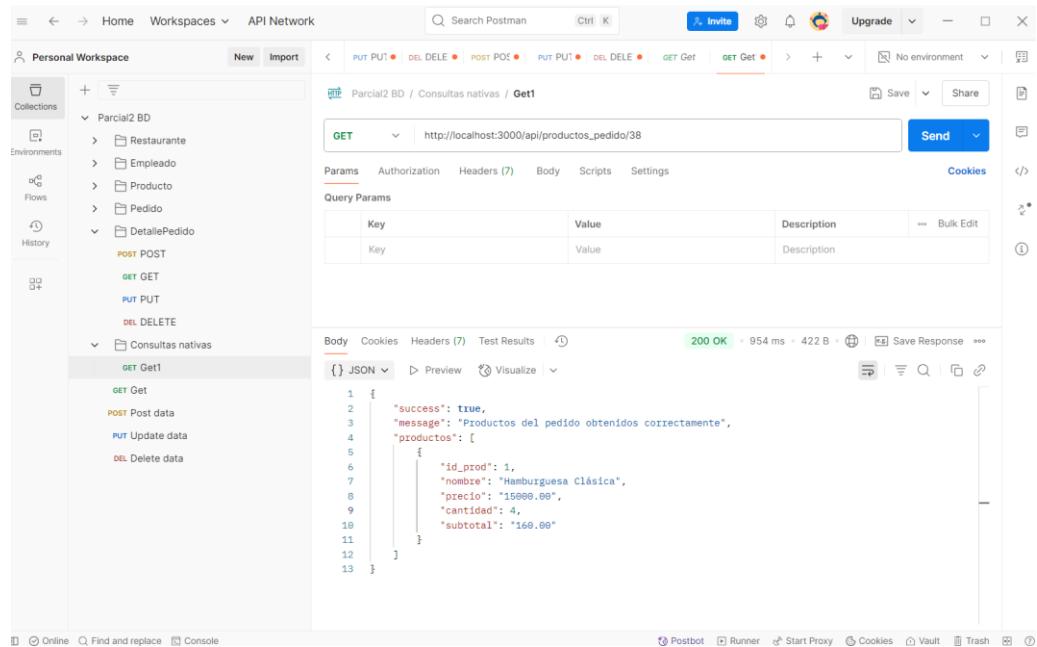
A través del código se obtienen todos los productos que están asociados a un pedido específico (por su id\_pedido), incluyendo información detallada

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 });

```

## Validación en Postman

The screenshot shows the Postman application interface. On the left, the sidebar displays collections like 'Parcial2 BD' and its sub-collections 'Restaurante', 'Empleado', 'Producto', 'Pedido', and 'DetallePedido'. Under 'DetallePedido', there are several requests: 'GET GET', 'PUT PUT', 'DEL DELETE', and 'Consultas nativas'. 'Consultas nativas' is expanded, showing 'GET Get1', 'GET Get2', 'GET Get3', 'GET Get4', 'GET Get5', 'GET Get', 'POST Post data', 'PUT Update data', and 'DEL Delete data'. 'GET Get2' is currently selected. In the main workspace, a GET request is configured to 'http://localhost:3000/api/productos\_mas\_vendidos'. The 'Body' tab shows a JSON response:

```

1  {
2    "success": true,
3    "message": "Productos más vendidos obtenidos correctamente",
4    "productos": [
5      {
6        "id_prod": 1,
7        "nombre": "Hamburguesa Clásica",
8        "total_vendidos": "47"
9      },
10     {
11       "id_prod": 3,
12       "nombre": "Hamburguesa Doble Carne",
13       "total_vendidos": "26"
14     },
15     {
16       "id_prod": 2,
17       "nombre": "Hamburguesa BBQ",
18       "total_vendidos": "25"
19     }
20   ]
21 }

```

### 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

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Personal Workspace' collections like 'Parcial2 BD', 'Restaurante', 'Empleado', 'Producto', 'Pedido', and 'DetallePedido'. The main area shows a GET request to 'http://localhost:3000/api/total\_ventas\_por\_restaurante'. The response body is a JSON array with four elements:

```

1 [
2   {
3     "id_rest": 9,
4     "nombre": "Sabor Caribeño",
5     "total_ventas": "280000.00"
6   },
7   {
8     "id_rest": 3,
9     "nombre": "Burger Point",
10    "total_ventas": "2920.00"
11  },
12  {
13    "id_rest": 1,
14    "nombre": "La Cabaña del Sabor",
15    "total_ventas": "37490.00"
16  },
17  {
18    "id_rest": 2,
19    "nombre": "Tacos El Patrón",
20    "total_ventas": "3240.00"
21  }
22 ]

```

#### 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 coincide 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

The screenshot shows the Postman application interface. On the left, there's a sidebar with collections like 'Parcial2 BD', 'Restaurante', 'Empleado', 'Producto', 'Pedido', and 'DetallePedido'. Under 'DetallePedido', there are several methods: POST, GET, PUT, and DELETE. The 'GET' method is selected, and its URL is set to `http://localhost:3000/api/pedidos_por_fecha?fecha=2025-04-07`. In the 'Query Params' section, there is a table with two rows: 'fecha' (checked, Value: 2025-04-07) and 'Key' (unchecked, Value: Key, Description: Description). Below the table, the 'Body' tab is selected, showing a JSON response:

```

1 [
2   {
3     "id_pedido": 7,
4     "fecha": "2025-04-07T05:00:00.000Z",
5     "id_rest": 1,
6     "total": "160.00"
7   }
8 ]

```

## 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 quieras 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

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Personal Workspace' containing collections like 'Parcial2 BD' and 'Consultas nativas'. Under 'Consultas nativas', 'GET Get5' is selected. The main panel shows a 'GET' request to 'http://localhost:3000/api/empleados\_por\_rol?rol=Cocinero&id\_rest=10'. In the 'Params' tab, 'rol' is set to 'Cocinero' and 'id\_rest' is set to '10'. The response status is '200 OK' with a response time of '791 ms'. The response body is a JSON array with one element:

```

1 [
2   {
3     "id_empleado": 46,
4     "nombre": "Alejandro Franco",
5     "rol": "Cocinero",
6     "id_rest": 10
7   }
8 ]

```