Parcial Bases De Datos

Michael Stiven Castillo Castillo Id 825947

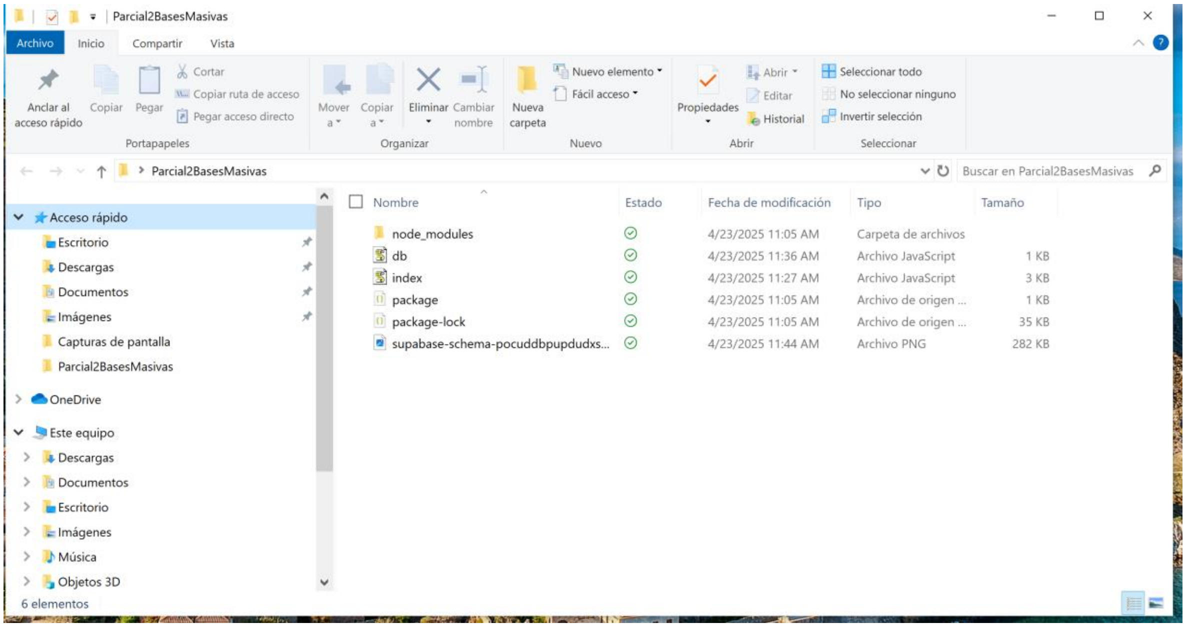
Corporación Universitaria Minuto de Dios

Ingeniería de Sistemas

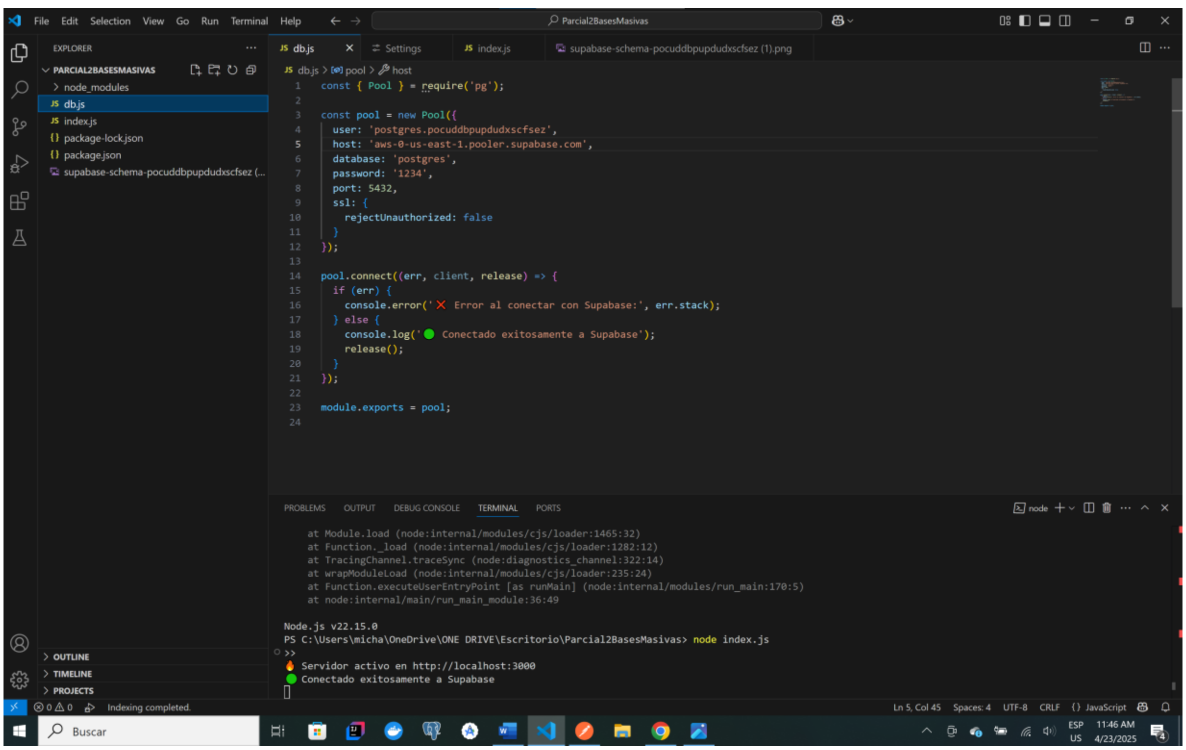
Bases de datos masivas

Alexander Matallana Porras

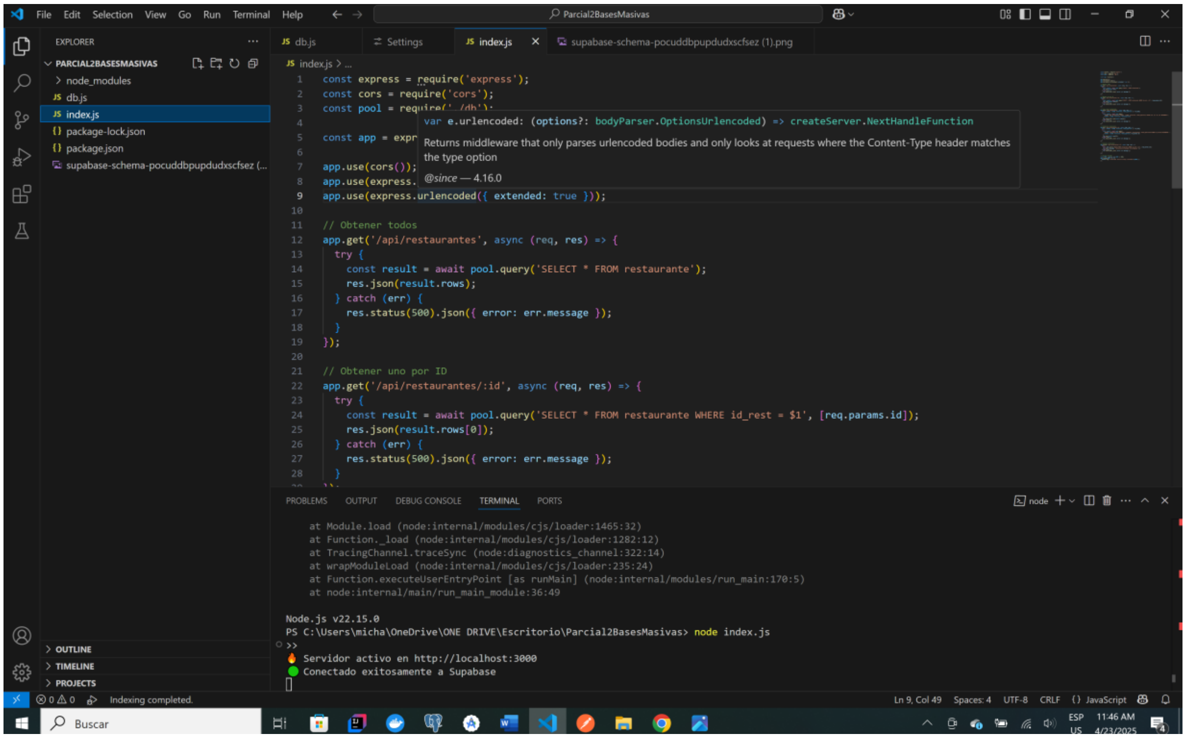
25 de abril del 2025

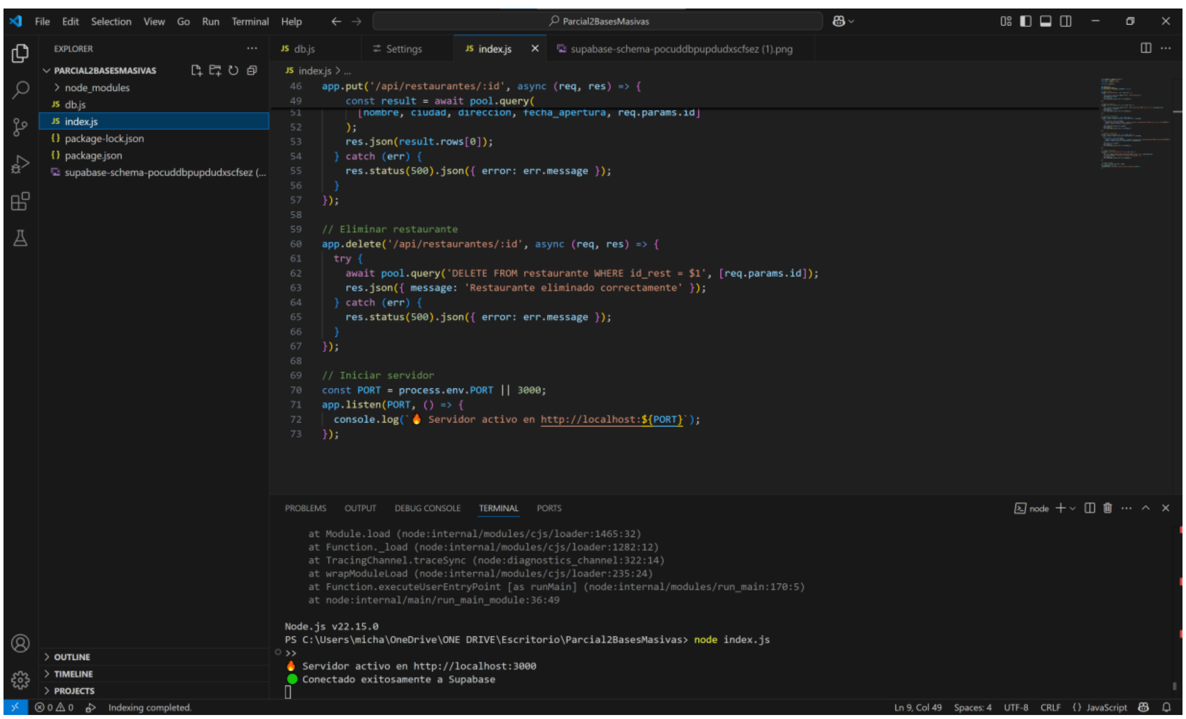


En el db.js va la conexión a base de datos PostgreSQL (Supabase)

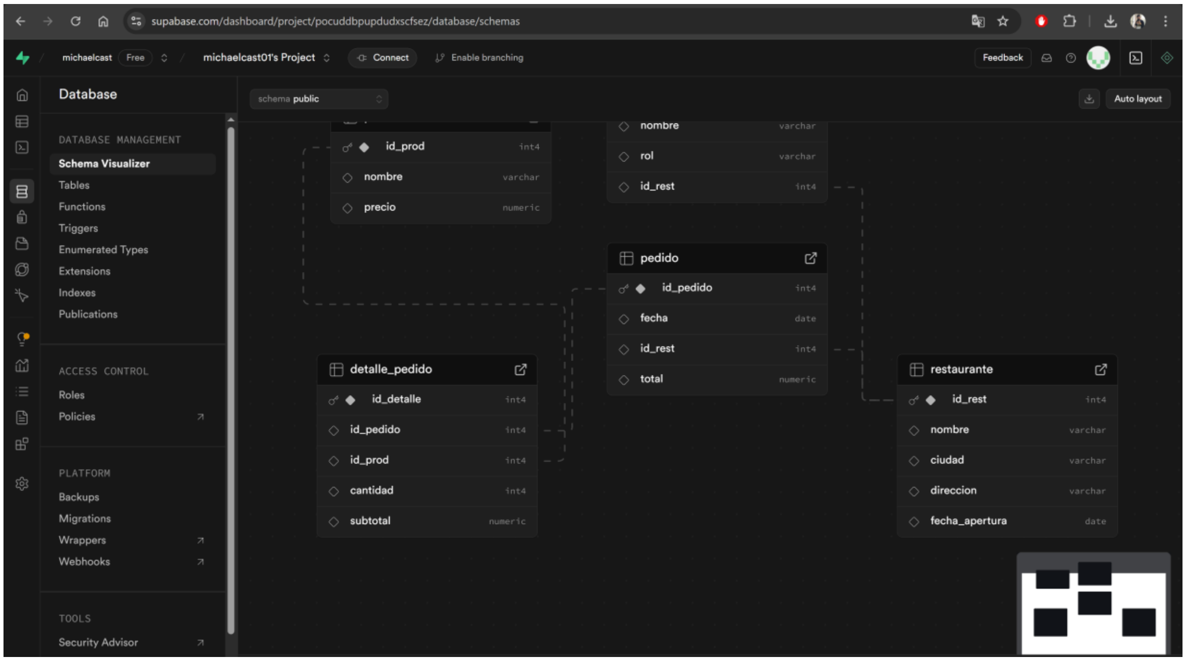


Crear rutas y controladores con Express para cada entidad

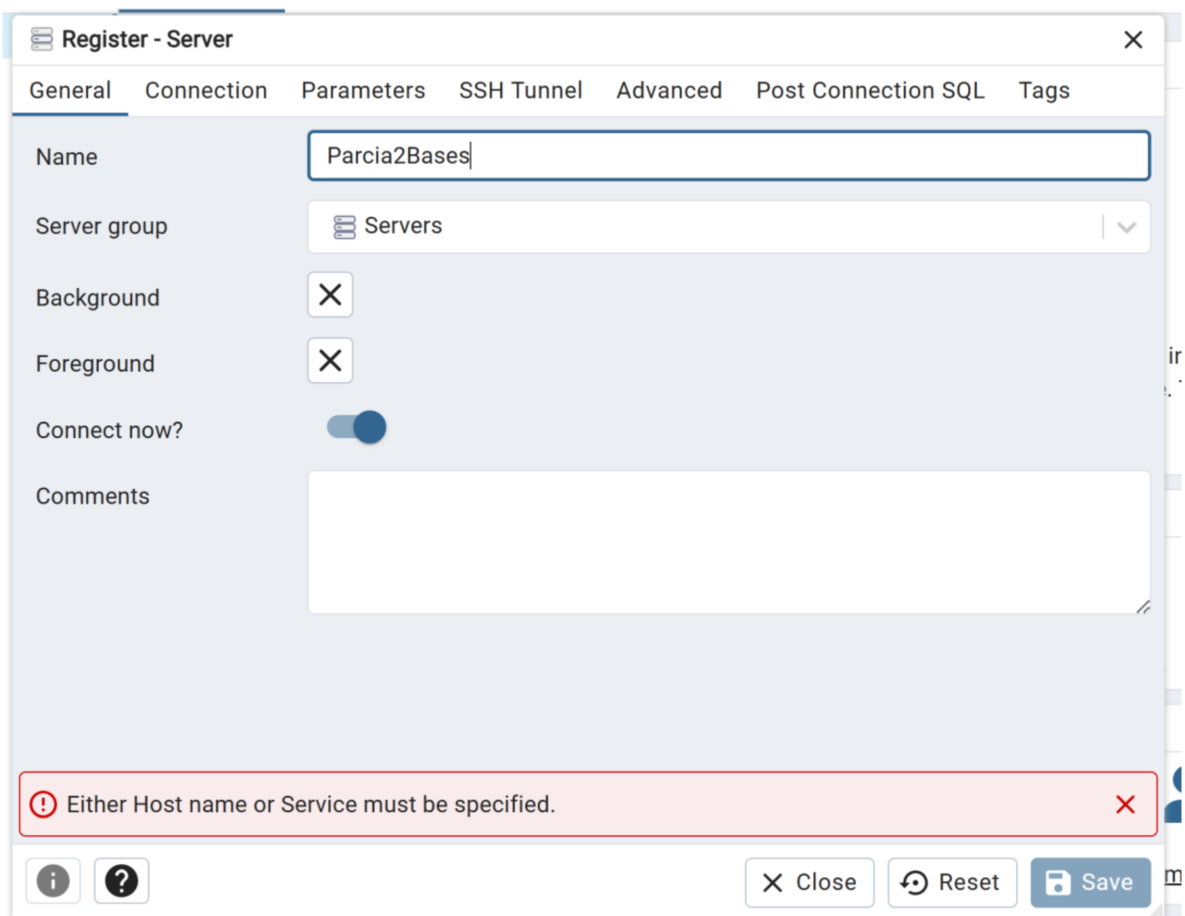


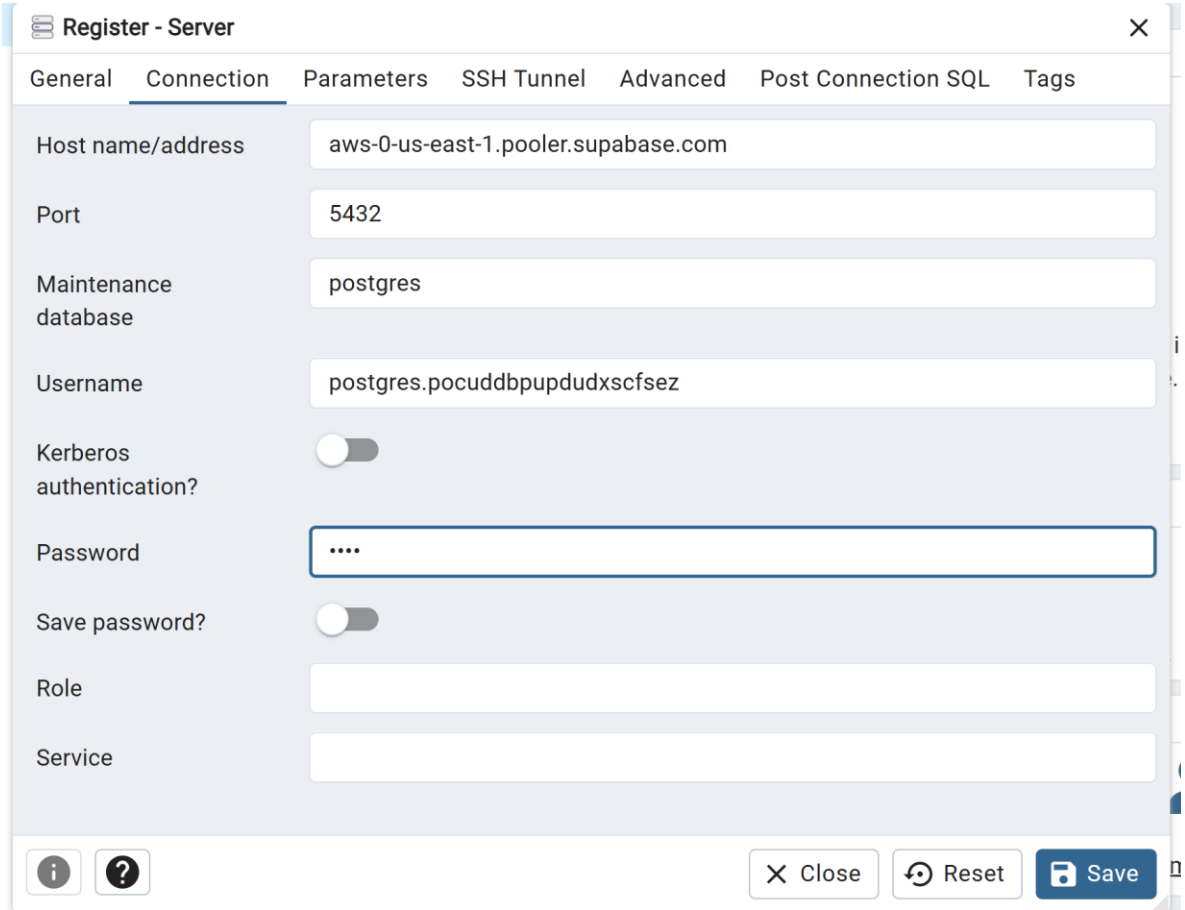


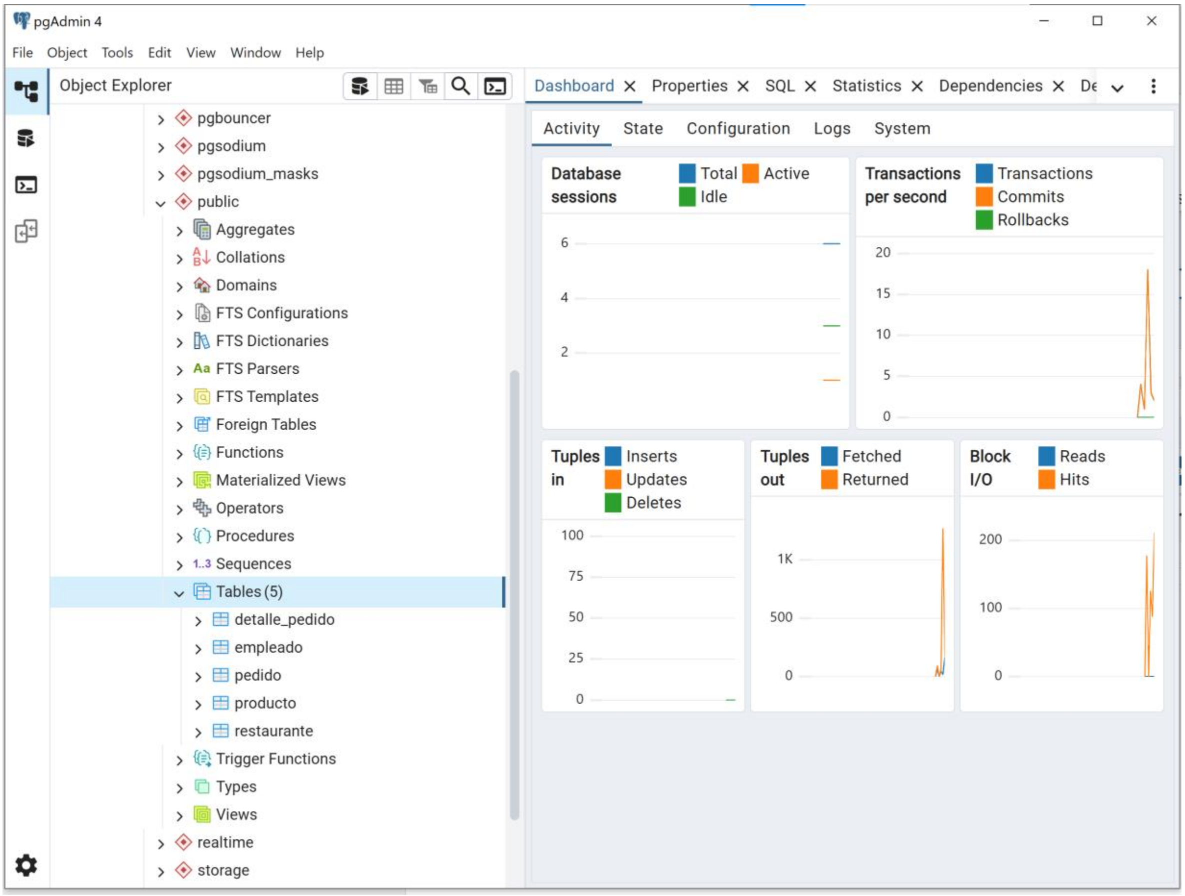
modelo de datos de Supabase



Crear la conexión con el pgadmin4




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



Consultas

1. Productos de un pedido específico

Muestra qué productos incluye un pedido, junto con la cantidad de cada uno y su subtotal. Sirve para ver el detalle completo de lo que pidió un cliente en un pedido.

2. Productos más vendidos (más de X unidades)

Te da una lista de productos que se han vendido más de X veces. Se calcula con la suma de las cantidades. Ideal para ver qué comidas o combos son los más populares.

3. Total de ventas por restaurante

Suma el total de todos los pedidos hechos en cada restaurante. Para hacer reportes de rendimiento de cada sucursal.

4. **Pedidos por fecha específica**

Filtra los pedidos que se hicieron en un día específico. Sirve para reportes diarios o buscar pedidos hechos en una fecha puntual.

5. **Empleados por rol en un restaurante**

Filtra empleados que trabajan en un restaurante por su rol (ej: cocineros, cajeros). Para ver la distribución del personal o buscar a alguien con un rol específico en una sucursal.

```
// Actualizar un pedido
app.put('/api/pedidos/:id', async (req, res) => {
  const { fecha, id_rest, total } = req.body;
  try {
    const result = await pool.query(
      'UPDATE pedido SET fecha = $1, id_rest = $2, total = $3 WHERE id_pedido = $4 RETURNING *',
      [fecha, id_rest, total, req.params.id]
    );
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Eliminar restaurante
app.delete('/api/restaurantes/:id', async (req, res) => {
  try {
    await pool.query('DELETE FROM restaurante WHERE id_rest = $1', [req.params.id]);
    res.json({ message: 'Restaurante eliminado correctamente' });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

```
//  Obtener todos los empleados
app.get('/api/empleados', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM empleado');
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

//  Obtener un empleado por ID
app.get('/api/empleados/:id', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM empleado WHERE id_empleado = $1', [req.params.id]);
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

```javascript
//  Crear nuevo empleado
app.post('/api/empleados', async (req, res) => {
  const { nombre, rol, id_rest } = req.body;
  try {
    const result = await pool.query(
      'INSERT INTO empleado (nombre, rol, id_rest) VALUES ($1, $2, $3) RETURNING *',
      [nombre, rol, id_rest]
    );
    res.status(201).json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

//  Actualizar empleado
app.put('/api/empleados/:id', async (req, res) => {
  const { nombre, rol, id_rest } = req.body;
  try {
    const result = await pool.query(
      'UPDATE empleado SET nombre = $1, rol = $2, id_rest = $3 WHERE id_empleado = $4 RETURNING *',
      [nombre, rol, id_rest, req.params.id]
    );
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

```javascript
//  Eliminar empleado
app.delete('/api/empleados/:id', async (req, res) => {
  try {
    await pool.query('DELETE FROM empleado WHERE id_empleado = $1', [req.params.id]);
    res.json({ message: 'Empleado eliminado correctamente' });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
// Obtener todos los productos
app.get('/api/productos', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM producto');
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Obtener un producto por ID
app.get('/api/productos/:id', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM producto WHERE id_prod = $1', [req.params.id]);
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

```javascript
// Crear producto
app.post('/api/productos', async (req, res) => {
  const { nombre, precio } = req.body;
  try {
    const result = await pool.query(
      'INSERT INTO producto (nombre, precio) VALUES ($1, $2) RETURNING *',
      [nombre, precio]
    );
    res.status(201).json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Actualizar producto
app.put('/api/productos/:id', async (req, res) => {
  const { nombre, precio } = req.body;
  try {
    const result = await pool.query(
      'UPDATE producto SET nombre = $1, precio = $2 WHERE id_prod = $3 RETURNING *',
      [nombre, precio, req.params.id]
    );
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

```javascript
// Eliminar producto
app.delete('/api/productos/:id', async (req, res) => {
  try {
    await pool.query('DELETE FROM producto WHERE id_prod = $1', [req.params.id]);
    res.json({ message: 'Producto eliminado correctamente' });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
// Obtener todos los pedidos
app.get('/api/pedidos', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM pedido');
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Obtener un pedido por ID
app.get('/api/pedidos/:id', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM pedido WHERE id_pedido = $1', [req.params.id]);
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

```javascript
// Crear nuevo pedido
app.post('/api/pedidos', async (req, res) => {
  const { fecha, id_rest, total } = req.body;
  try {
    const result = await pool.query(
      'INSERT INTO pedido (fecha, id_rest, total) VALUES ($1, $2, $3) RETURNING *',
      [fecha, id_rest, total]
    );
    res.status(201).json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Eliminar pedido
app.delete('/api/pedidos/:id', async (req, res) => {
  try {
    await pool.query('DELETE FROM pedido WHERE id_pedido = $1', [req.params.id]);
    res.json({ message: 'Pedido eliminado correctamente' });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

```javascript
// Obtener todos los detalles
app.get('/api/detalles', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM detalle_pedido');
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Obtener un detalle por ID
app.get('/api/detalles/:id', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM detalle_pedido WHERE id_detalle = $1', [req.params.id]);
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

```javascript
// Crear nuevo detalle
app.post('/api/detalles', async (req, res) => {
  const { id_pedido, id_prod, cantidad, subtotal } = req.body;
  try {
    const result = await pool.query(
      'INSERT INTO detalle_pedido (id_pedido, id_prod, cantidad, subtotal) VALUES ($1, $2, $3, $4) RETURNING *',
      [id_pedido, id_prod, cantidad, subtotal]
    );
    res.status(201).json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Actualizar detalle
app.put('/api/detalles/:id', async (req, res) => {
  const { id_pedido, id_prod, cantidad, subtotal } = req.body;
  try {
    const result = await pool.query(
      'UPDATE detalle_pedido SET id_pedido = $1, id_prod = $2, cantidad = $3, subtotal = $4 WHERE id_detalle = $5 RETURNING *',
      [id_pedido, id_prod, cantidad, subtotal, req.params.id]
    );
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

```javascript
// Eliminar detalle
app.delete('/api/detalles/:id', async (req, res) => {
  try {
    await pool.query('DELETE FROM detalle_pedido WHERE id_detalle = $1', [req.params.id]);
    res.json({ message: 'Detalle eliminado correctamente' });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

process.on('uncaughtException', (err) => {
  console.error('🔥 Error no capturado:', err);
});

process.on('unhandledRejection', (reason, promise) => {
  console.error('💥 Rechazo no manejado:', reason);
});
// Iniciar servidor
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`🔥 Servidor activo en http://localhost:${PORT}`);
});
```

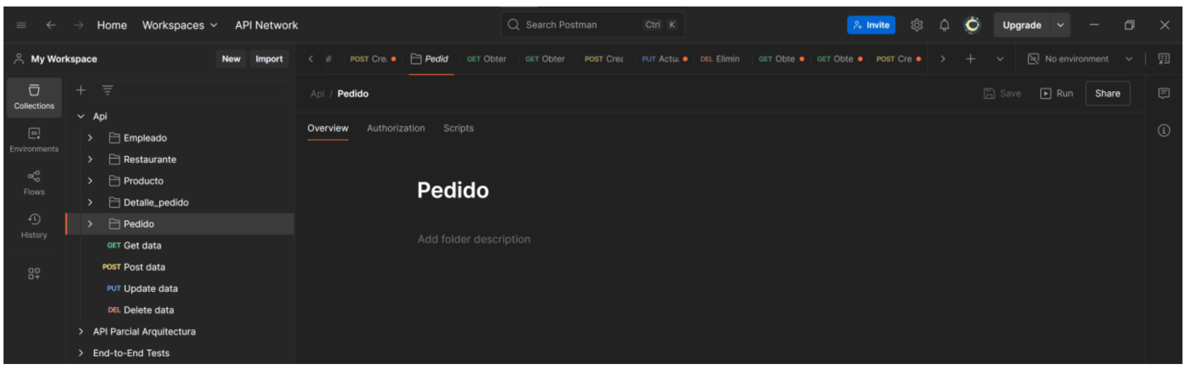
Ambos son eventos del proceso global de Node.js (process) que se encargan de atrapar errores graves que de otro modo harían que tu aplicación se caiga sin explicación. Son como redes de seguridad.

- uncaughtException: Errores no atrapados con try/catch = Evita que la app se cierre inesperadamente
- unhandledRejection: Promesas rechazadas sin .catch() = Detecta errores silenciosos en promesas

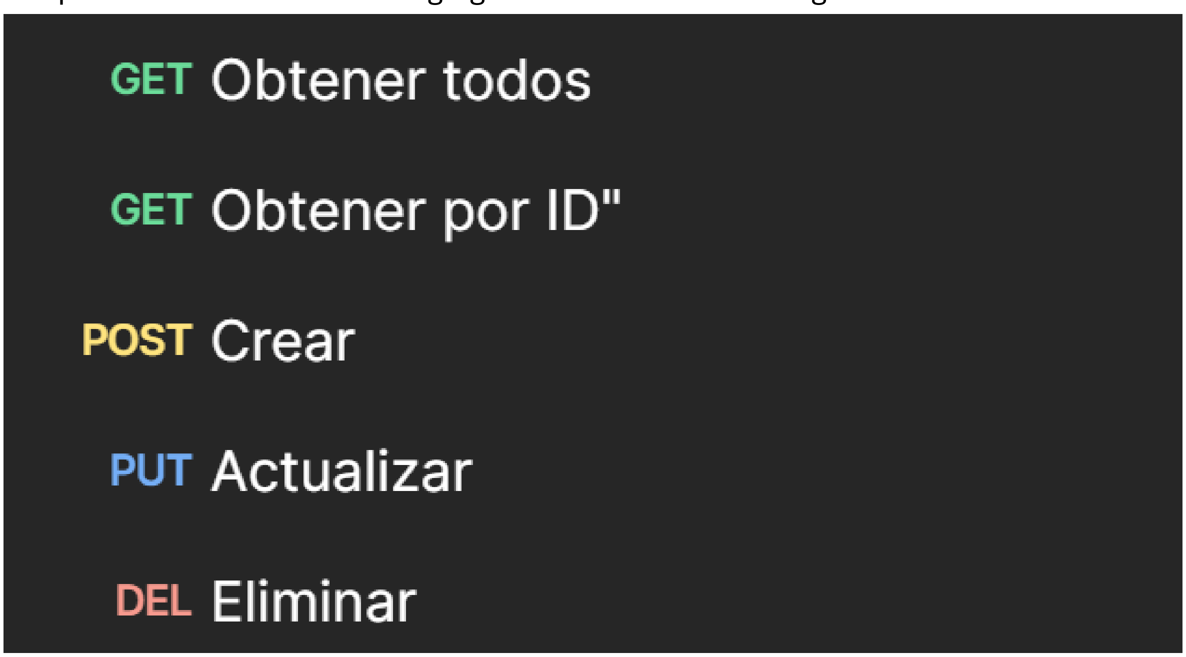
```
process.on('uncaughtException', (err) => {
  console.error('🔥 Error no capturado:', err);
});

process.on('unhandledRejection', (reason, promise) => {
  console.error('💥 Rechazo no manejado:', reason);
});
// Iniciar servidor
```

Ingresamos a postman y creamos una nueva collection en este caso se va a llamar Api

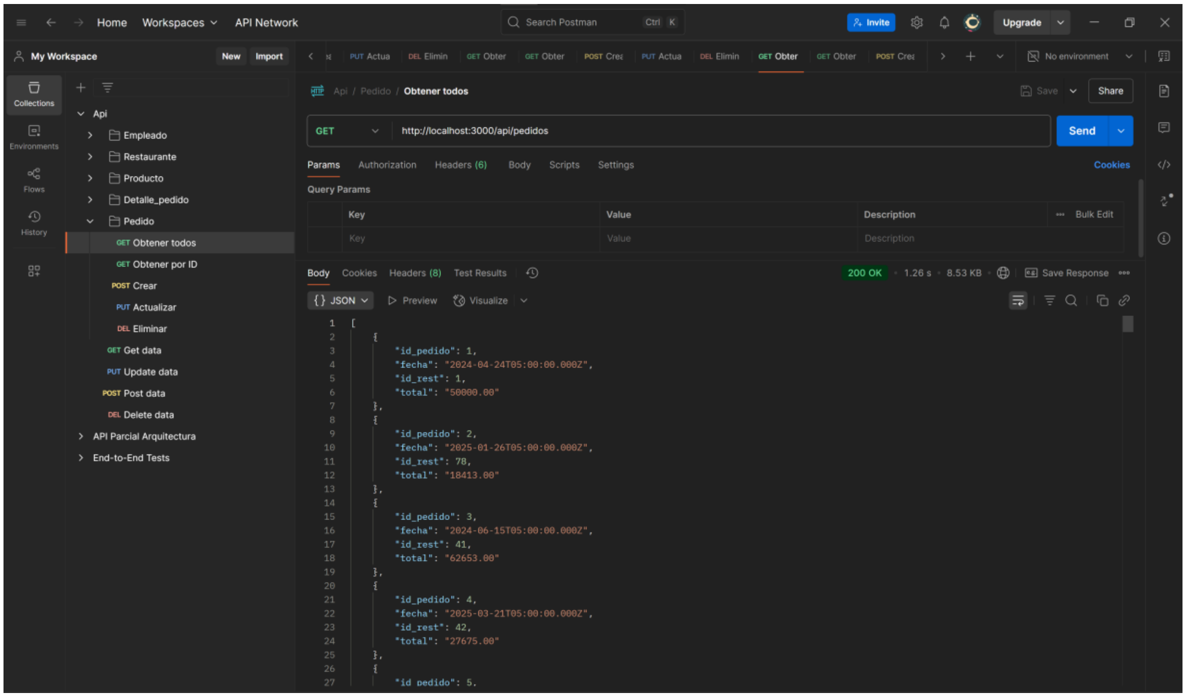
Dentro de Api debemos de crear 5 folder los cuales corresponde a las tablas del Ejercicio propuesto en clase



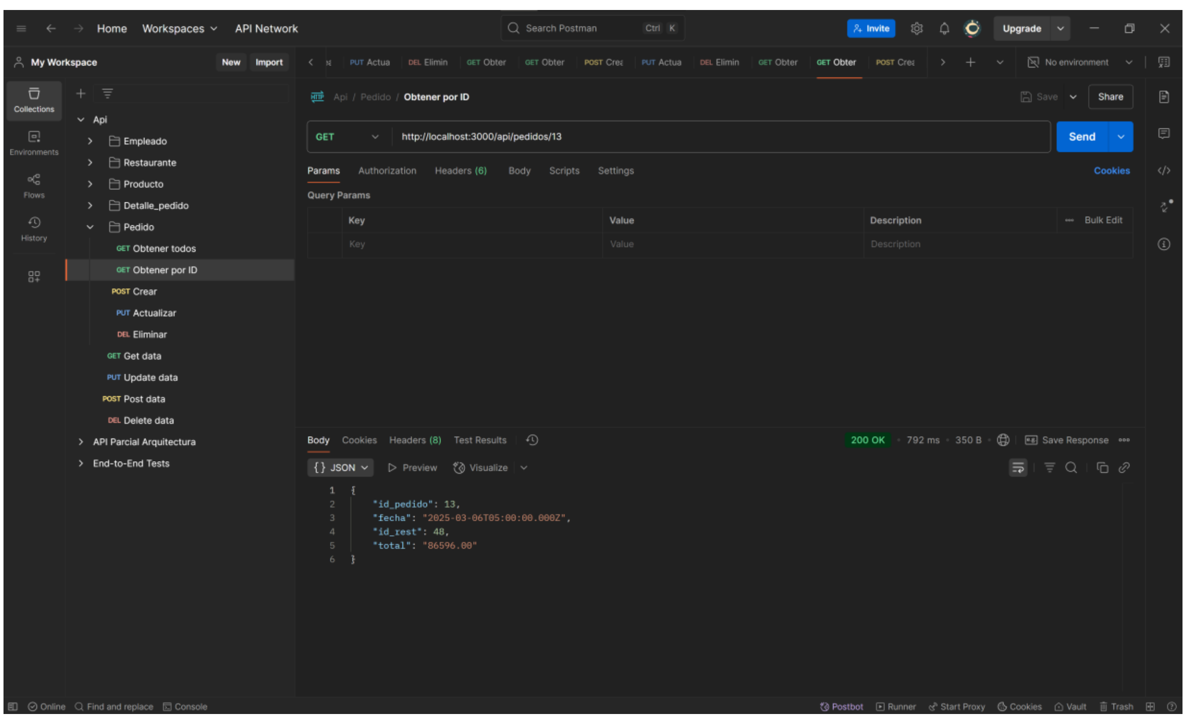
Despues de crear los 5 folder agregamos a cada folder lo siguiente



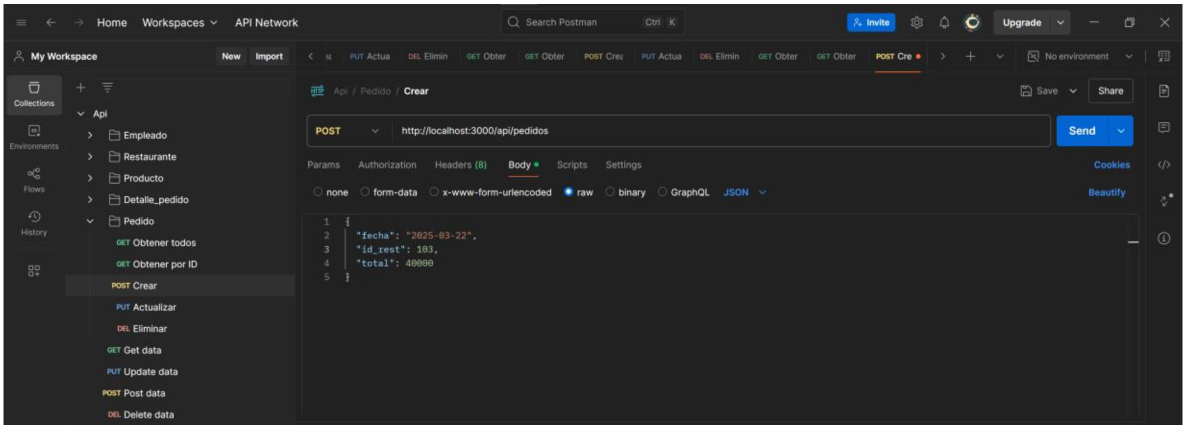
Nos muestra todos los pedidos



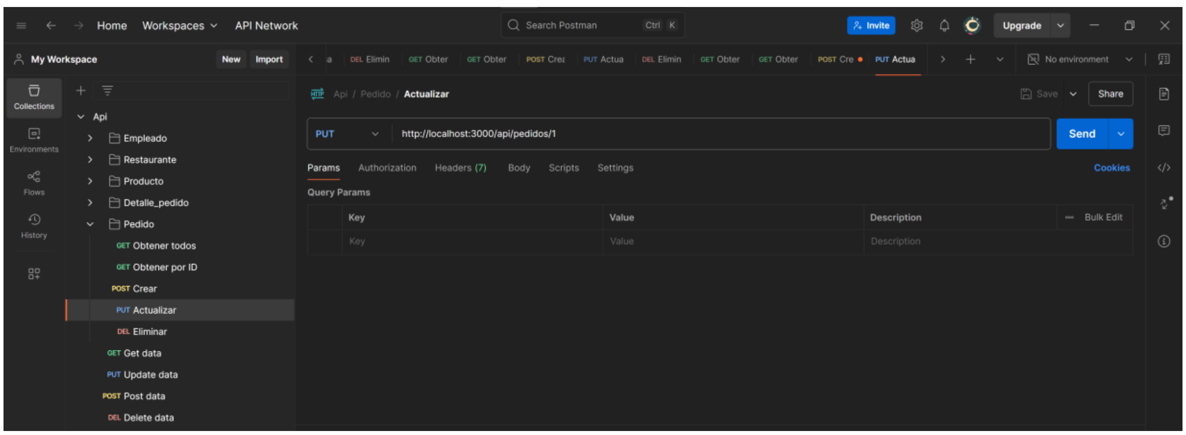
Nos muestra uno en específicos esto lo hace a través del id en etse caso como id_pedido



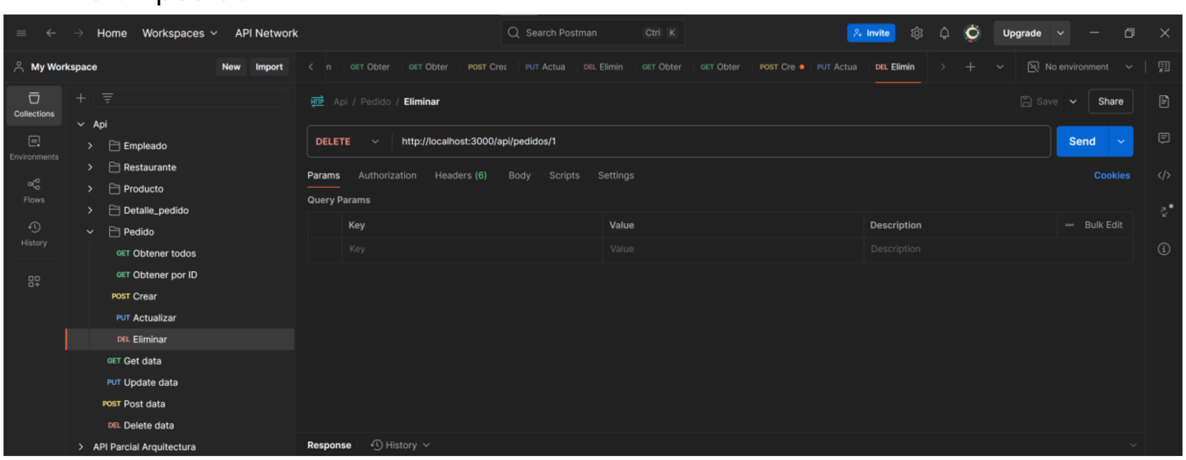
Crea un pedido



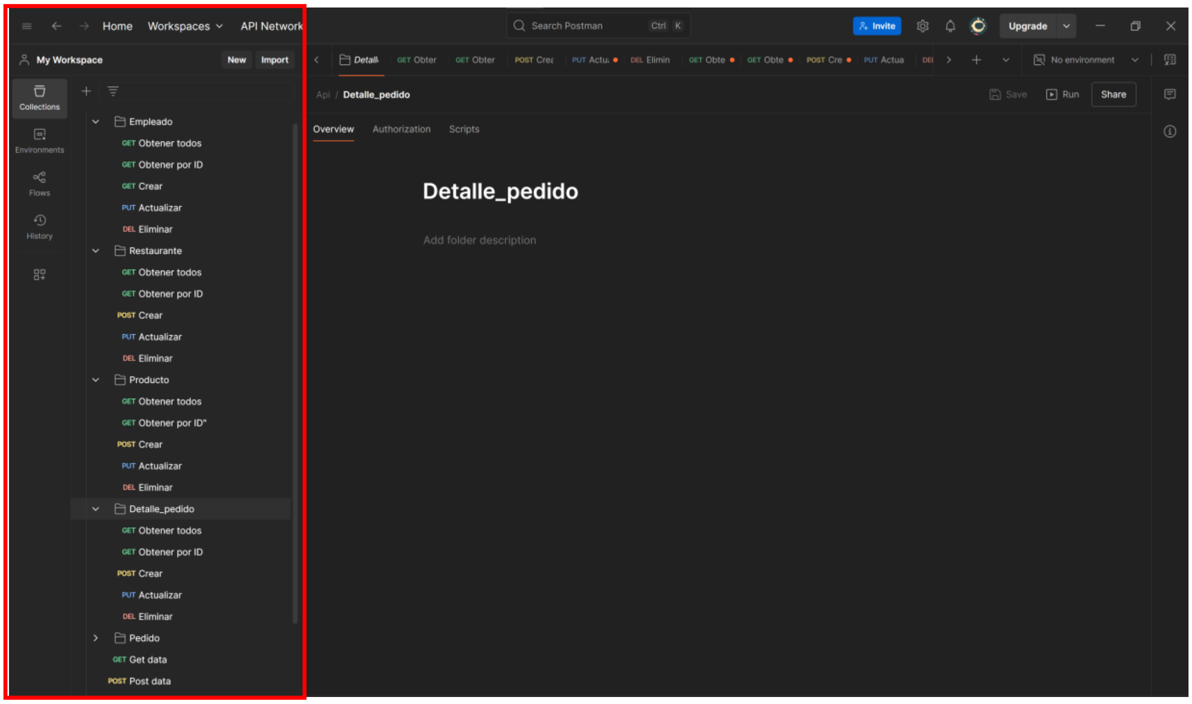
Actualiza un pedido



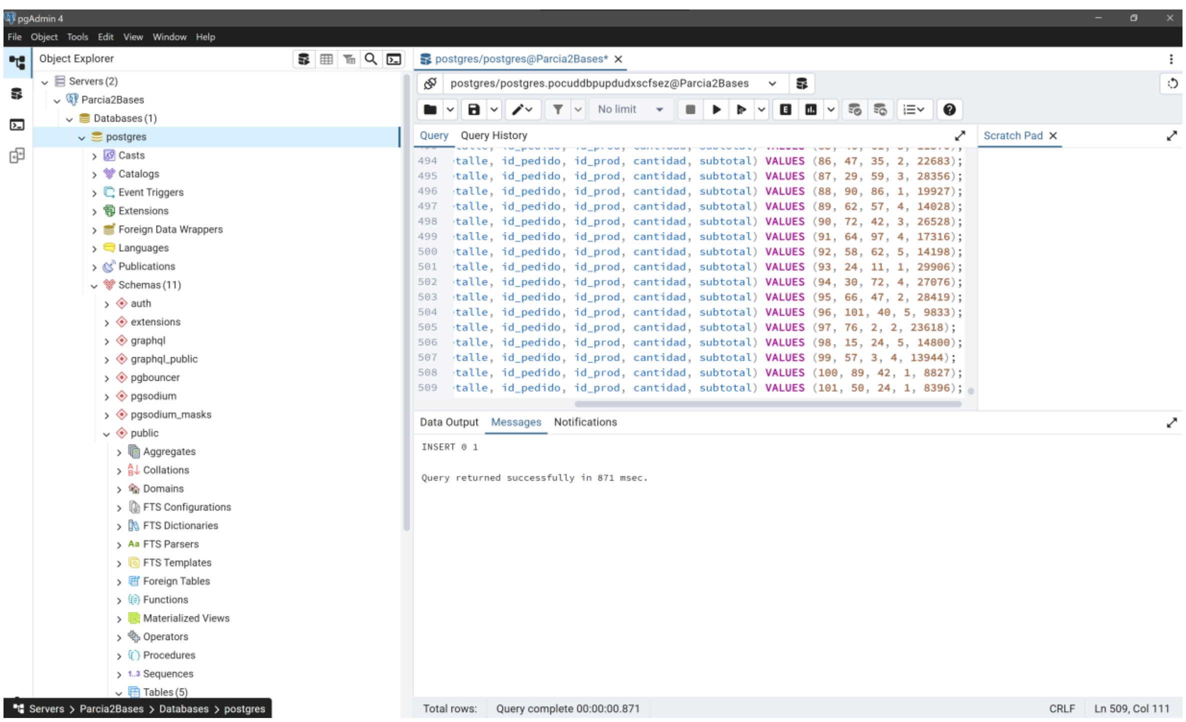
Elimina un pedido



Esto lo hacemos igual en los otros folder con el id correspondiente a cada tabla

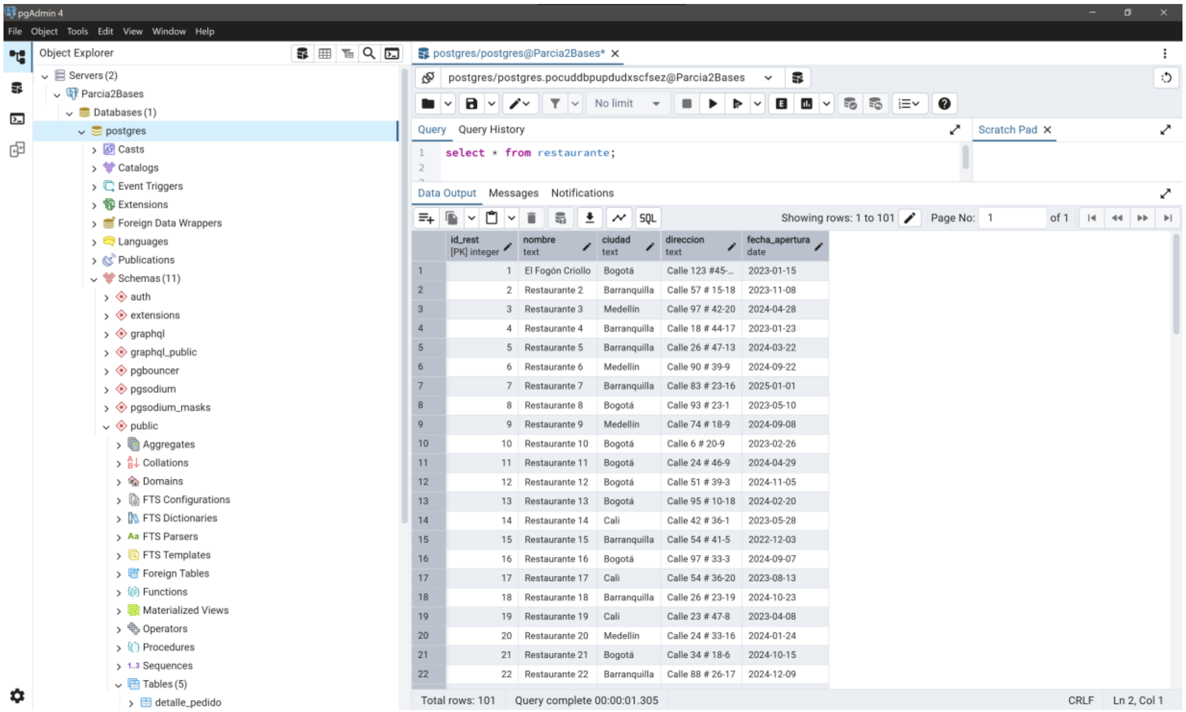


Ingresamos 100 registros por tabla

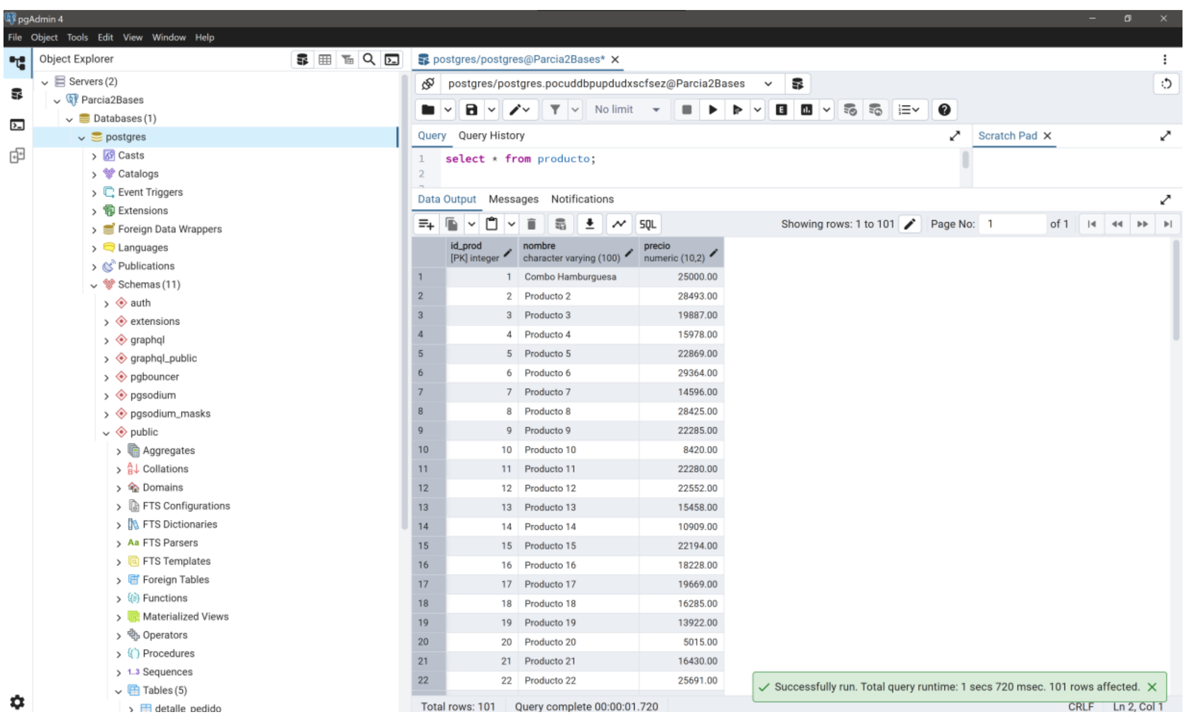


Ahora visualizamos los datos de cada tabla

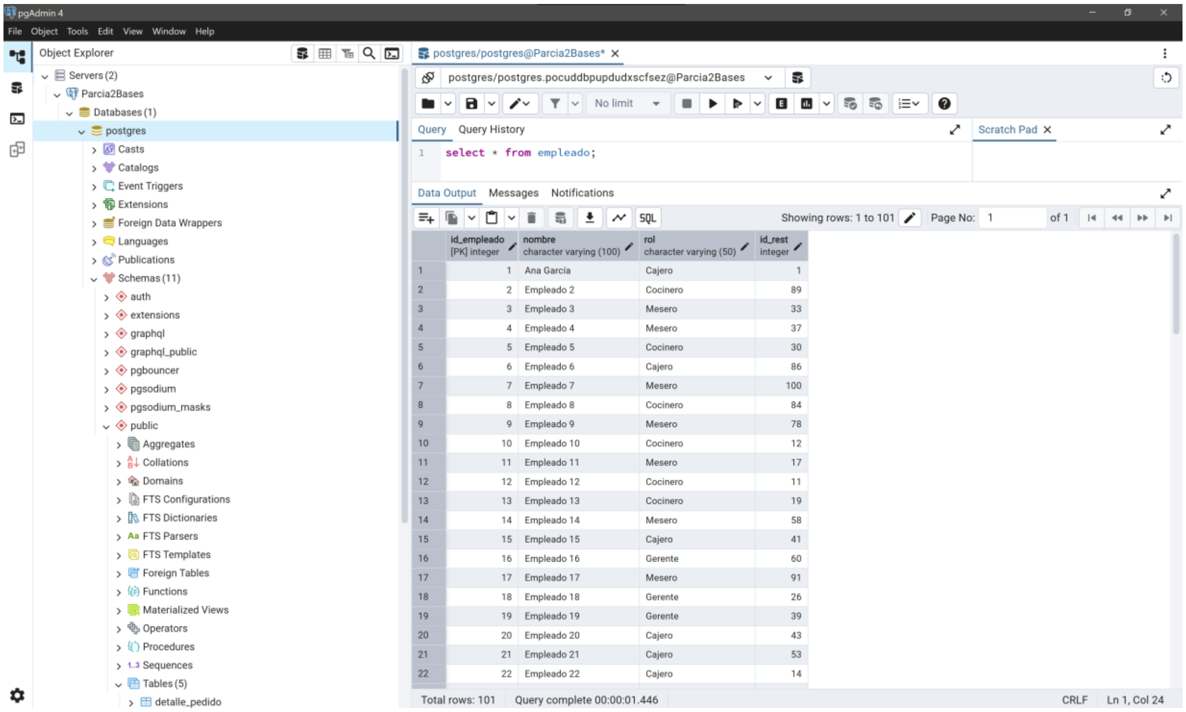
select * from restaurante;



select * from producto;



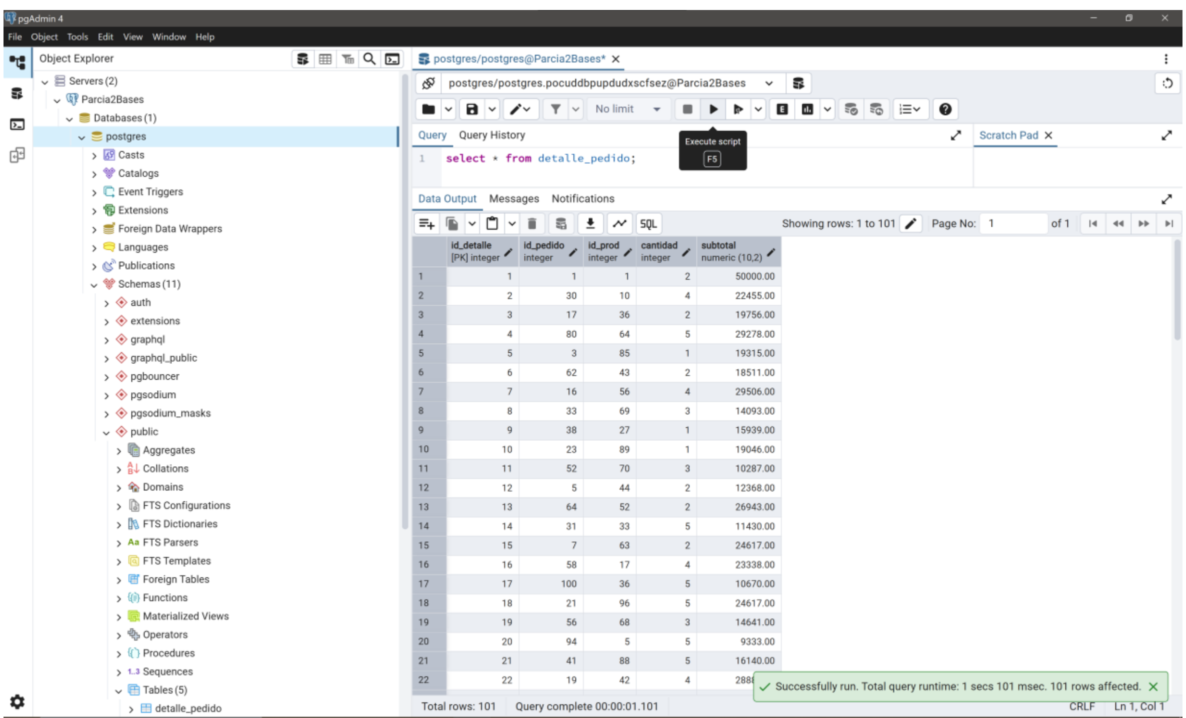
select * from empleado;



	id_empleado [PK] integer	nombre character varying (100)	rol character varying (50)	id_rest integer
1	1	Ana García	Cajero	1
2	2	Empleado 2	Cocinero	89
3	3	Empleado 3	Mesero	33
4	4	Empleado 4	Mesero	37
5	5	Empleado 5	Cocinero	30
6	6	Empleado 6	Cajero	86
7	7	Empleado 7	Mesero	100
8	8	Empleado 8	Cocinero	84
9	9	Empleado 9	Mesero	78
10	10	Empleado 10	Cocinero	12
11	11	Empleado 11	Mesero	17
12	12	Empleado 12	Cocinero	11
13	13	Empleado 13	Cocinero	19
14	14	Empleado 14	Mesero	58
15	15	Empleado 15	Cajero	41
16	16	Empleado 16	Gerente	60
17	17	Empleado 17	Mesero	91
18	18	Empleado 18	Gerente	26
19	19	Empleado 19	Gerente	39
20	20	Empleado 20	Cajero	43
21	21	Empleado 21	Cajero	53
22	22	Empleado 22	Cajero	14

Total rows: 101 Query complete 00:00:01.446

select * from detalle_pedido;

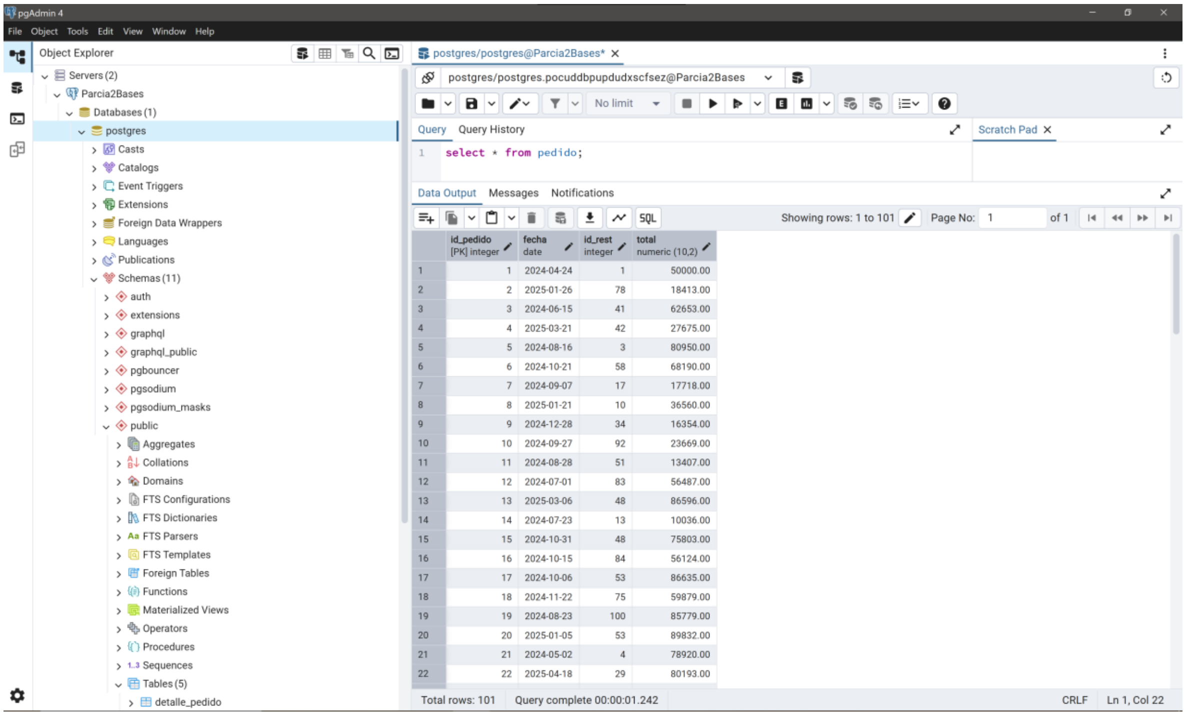


	id_detalle [PK] integer	id_pedido integer	id_prod integer	cantidad integer	subtotal numeric (10,2)
1	1	1	1	2	50000.00
2	2	30	10	4	22455.00
3	3	17	36	2	19756.00
4	4	80	64	5	29278.00
5	5	3	85	1	19315.00
6	6	62	43	2	18511.00
7	7	16	56	4	29506.00
8	8	33	69	3	14093.00
9	9	38	27	1	15939.00
10	10	23	89	1	19046.00
11	11	52	70	3	10287.00
12	12	5	44	2	12368.00
13	13	64	52	2	26943.00
14	14	31	33	5	11430.00
15	15	7	63	2	24617.00
16	16	58	17	4	23338.00
17	17	100	36	5	10670.00
18	18	21	96	5	24617.00
19	19	56	68	3	14641.00
20	20	94	5	5	9333.00
21	21	41	88	5	16140.00
22	22	19	42	4	[illegible]

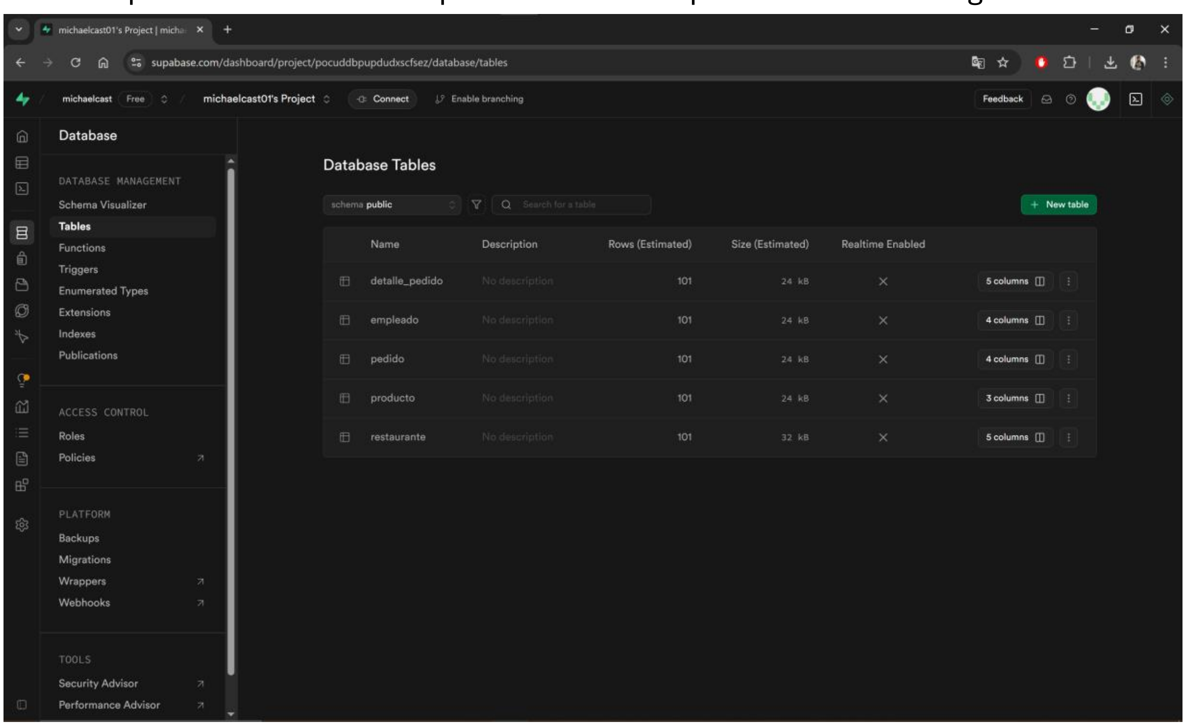
Successfully run. Total query runtime: 1 secs 101 msec. 101 rows affected.

Total rows: 101 Query complete 00:00:01.101

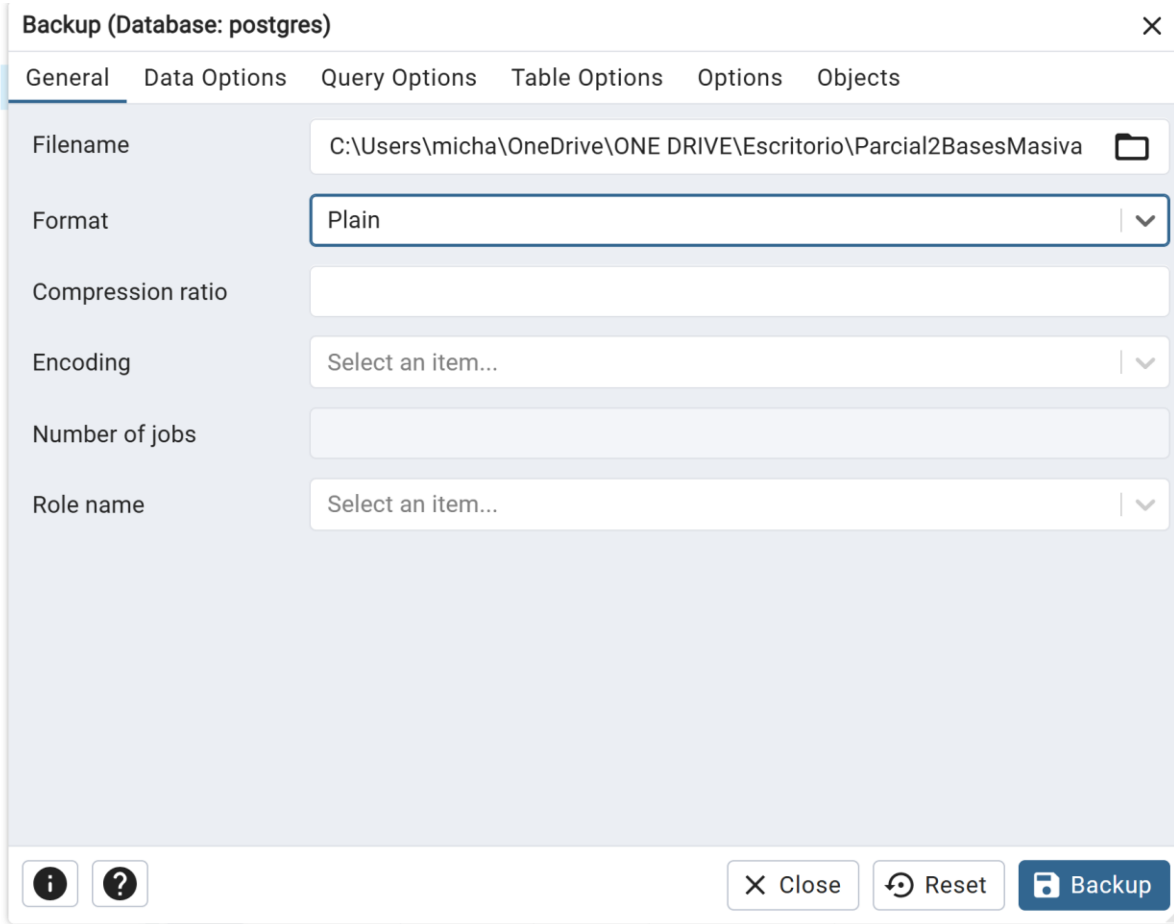
select * from pedido;

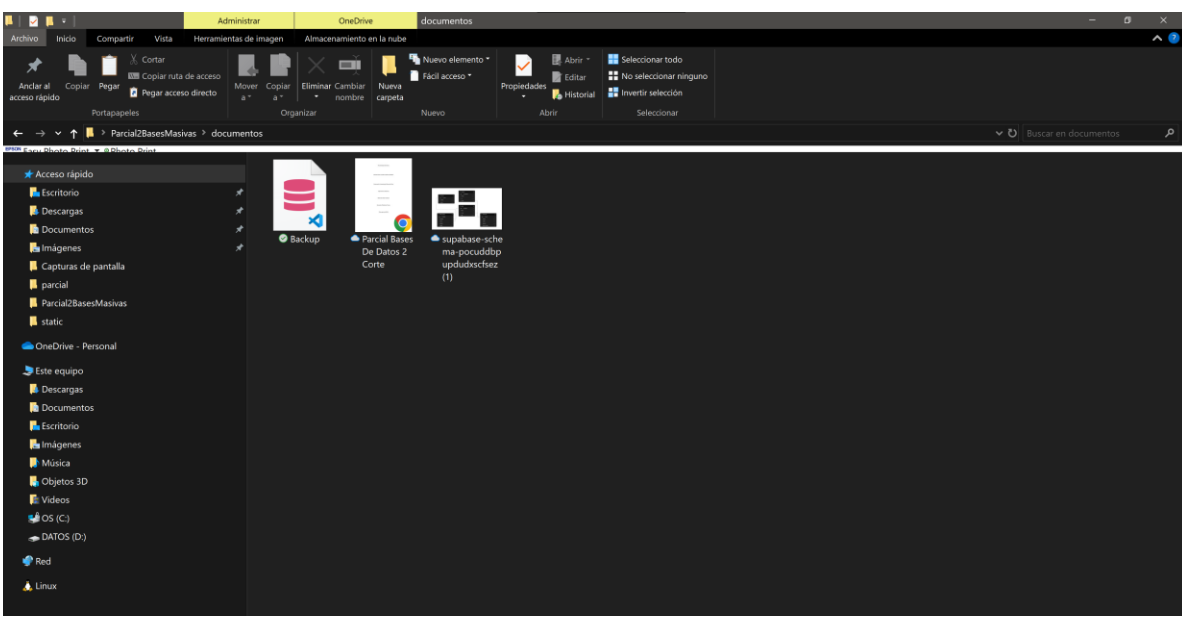


También podemos validar en supabase los datos que anteriormente ingresamos



Ahora procedemos hacer el backup de pgadmin4
debemos de ingresar la ruta donde se va a guardar el backup generado





Ahora exportamos la collection de postman