

Parcial Bases De Datos

Michael Stiven Castillo Castillo Id 825947

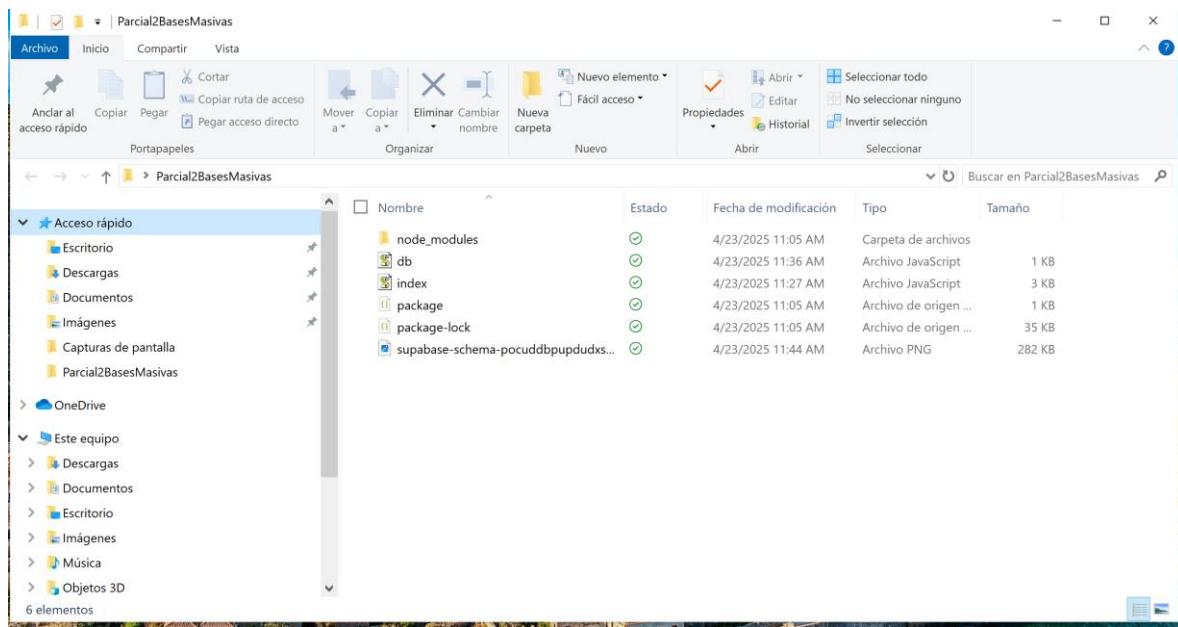
Corporación Universitaria Minuto de Dios

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)

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows a project structure with files like `index.js`, `db.js`, and `package.json`.
- Code Editor:** The `db.js` file is open, displaying code for connecting to a PostgreSQL database via Supabase's pooler. The code uses the `pg` module to create a pool and handle connections.
- Terminal:** The terminal shows the Node.js environment (`v22.15.0`) and the command `node index.js`. It outputs "Servidor activo en http://localhost:3000" and "Conectado exitosamente a Supabase".
- Status Bar:** Shows the current file is `index.js`, with 5 lines of code, 4 spaces, and CRLF line endings. It also shows the date and time as 4/23/2025 at 11:46 AM.

Crear rutas y controladores con Express para cada entidad

```
File Edit Selection View Go Run Terminal Help ⏎ → Parcial2BasesMasivas
```

EXPLORER JS db.js Settings JS index.js supabase-schema-pocuddbpupudscfsez (1).png

index.js > ...

```
1 const express = require('express');
2 const cors = require('cors');
3 const pool = require('./db');
4 var e.urlencoded: (options?: bodyParser.OptionsUrlencoded) => createServer.NextHandleFunction
5 const app = expr Returns middleware that only parses urlencoded bodies and only looks at requests where the Content-Type header matches
6 the type option
7 app.use(cors());
8 app.use(express. @since -4.16.0
9 app.use(express.urlencoded({ extended: true }));
```

// Obtener todos

```
10 app.get('/api/restaurantes', async (req, res) => {
11   try {
12     const result = await pool.query("SELECT * FROM restaurante");
13     res.json(result.rows);
14   } catch (err) {
15     res.status(500).json({ error: err.message });
16   }
17 });
18 });
19 });

// Obtener uno por ID
20 app.get('/api/restaurantes/:id', async (req, res) => {
21   try {
22     const result = await pool.query('SELECT * FROM restaurante WHERE id_rest = $1', [req.params.id]);
23     res.json(result.rows[0]);
24   } catch (err) {
25     res.status(500).json({ error: err.message });
26   }
27 });
28 });

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

at Module.load (node:internal/modules/cjs/loader:1465:32)
at Function._load (node:internal/modules/cjs/loader:1282:12)
at TracingChannel.traceSync (node:diagnostics_channel:322:14)
at wrapModuleLoad (node:internal/modules/cjs/loader:235:24)
at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:170:5)
at node:internal/main/run_main_module:36:49

Node.js v22.15.0
PS C:\Users\micha\OneDrive\One DRIVE\Escritorio\Parcial2BasesMasivas> node index.js

>>> 🔥 Servidor activo en http://localhost:3000
● Conectado exitosamente a Supabase

Indexing completed.

Buscar

In 9, Col 49 Spaces: 4 UTF-8 CRLF () JavaScript ESP 11:46 AM VS 4/23/2024

```
File Edit Selection View Go Run Terminal Help ⏎ → Parcial2BasesMasivas
```

EXPLORER JS db.js Settings JS index.js supabase-schema-pocuddbpupudscfsez (1).png

index.js > ...

```
46 app.put('/api/restaurantes/:id', async (req, res) => {
47   const result = await pool.query(
48     [nombre, ciudad, direccion, fecha_apertura, req.params.id]
49   );
50   res.json(result.rows[0]);
51 } catch (err) {
52   res.status(500).json({ error: err.message });
53 }
54 });

// Eliminar restaurante
55 app.delete('/api/restaurantes/:id', async (req, res) => {
56   try {
57     await pool.query('DELETE FROM restaurante WHERE id_rest = $1', [req.params.id]);
58     res.json({ message: 'Restaurante eliminado correctamente' });
59   } catch (err) {
60     res.status(500).json({ error: err.message });
61   }
62 });
63 });

// Iniciar servidor
64 const PORT = process.env.PORT || 3000;
65 app.listen(PORT, () => {
66   console.log(`🔥 Servidor activo en http://localhost:${PORT}`);
67 });
68 });

const express = require('express');
const cors = require('cors');
const pool = require('./db');
const app = express();

app.use(cors());
app.use(express.urlencoded({ extended: true }));

app.get('/api/restaurantes', async (req, res) => {
  try {
    const result = await pool.query("SELECT * FROM restaurante");
    res.json(result.rows);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

app.get('/api/restaurantes/:id', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM restaurante WHERE id_rest = $1', [req.params.id]);
    res.json(result.rows[0]);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

app.put('/api/restaurantes/:id', async (req, res) => {
  const { nombre, ciudad, direccion, fecha_apertura } = req.body;
  const id = req.params.id;
  const result = await pool.query(
    `UPDATE restaurante SET nombre=$1, ciudad=$2, direccion=$3, fecha_apertura=$4 WHERE id_rest=$5`,
    [nombre, ciudad, direccion, fecha_apertura, id]
  );
  res.json(result.rows[0]);
});

app.delete('/api/restaurantes/:id', async (req, res) => {
  const id = req.params.id;
  const result = await pool.query('DELETE FROM restaurante WHERE id_rest = $1', [id]);
  res.json({ message: 'Restaurante eliminado correctamente' });
});

app.listen(3000, () => {
  console.log(`🔥 Servidor activo en http://localhost:3000`);
});
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

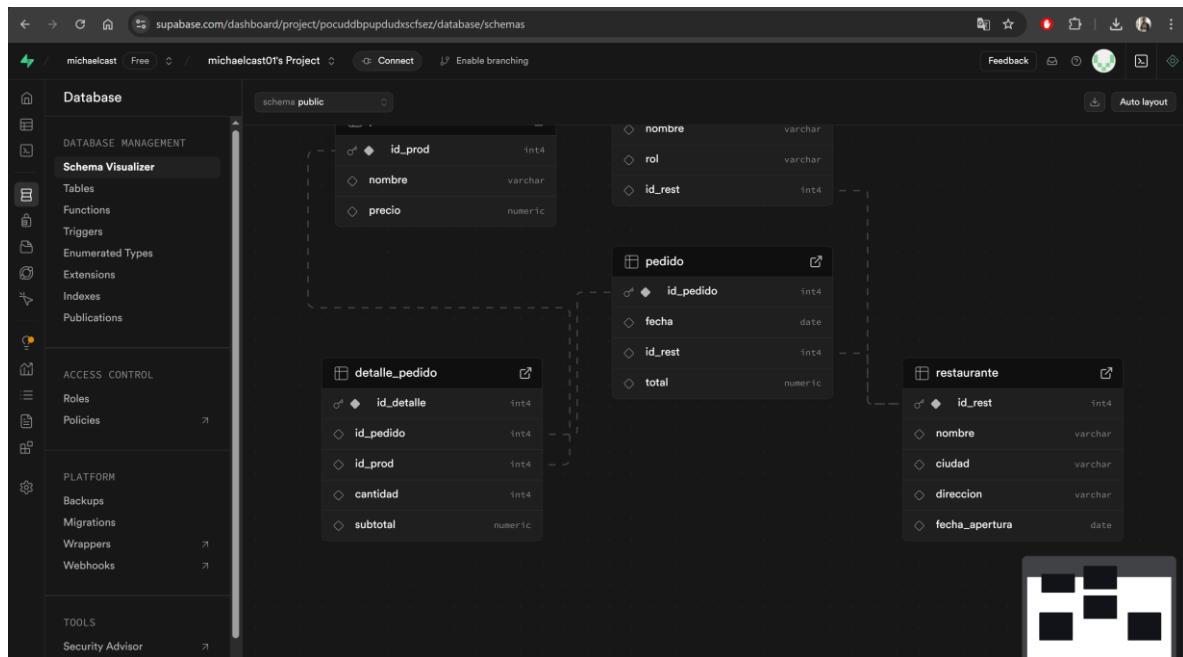
at Module.load (node:internal/modules/cjs/loader:1465:32)
at Function._load (node:internal/modules/cjs/loader:1282:12)
at TracingChannel.traceSync (node:diagnostics_channel:322:14)
at wrapModuleLoad (node:internal/modules/cjs/loader:235:24)
at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:170:5)
at node:internal/main/run_main_module:36:49

Node.js v22.15.0
PS C:\Users\micha\OneDrive\One DRIVE\Escritorio\Parcial2BasesMasivas> node index.js

>>> 🔥 Servidor activo en http://localhost:3000
● Conectado exitosamente a Supabase

Indexing completed.

modelo de datos de Supabase



Crear la conexión con el pgadmin4

Register - Server

General Connection Parameters SSH Tunnel Advanced Post Connection SQL Tags

Name	Parcia2Bases
Server group	Servers
Background	X
Foreground	X
Connect now?	<input checked="" type="checkbox"/>
Comments	

Either Host name or Service must be specified.

Buttons: Close, Reset, Save

Register - Server

X

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

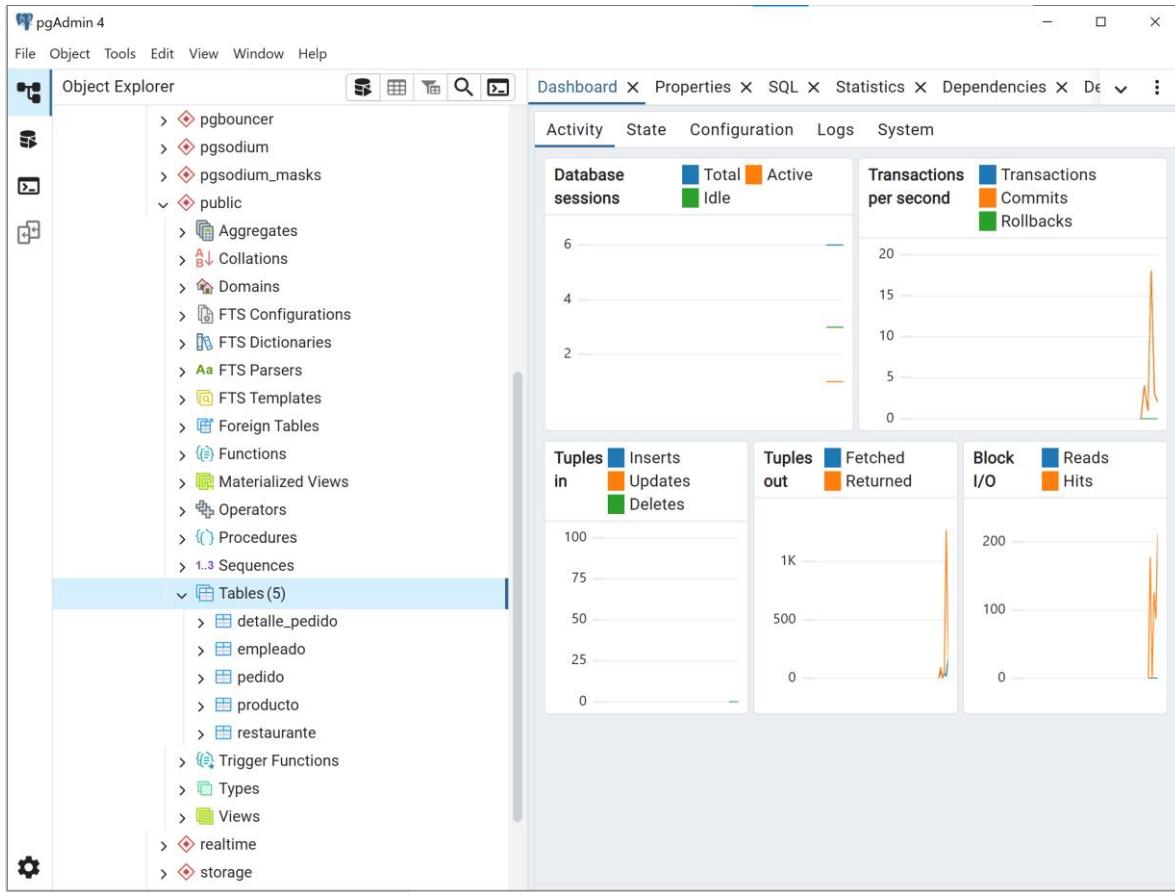
Service



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

Filtrar 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

Filtrar 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 });
  }
});
```

```

// 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 });
  }
});

```

```

// 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 });
  }
});

```

```

// 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 });
  }
});

```

```

// 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 });
  }
});

```

```

// 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 });
  }
});

```

```

// 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 });
  }
});

```

```

// 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 });
  }
});

// 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);
});

// Inician servidores

```

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

The screenshot shows the Postman interface with a collection named 'Api' selected. Under the 'Pedido' folder, the 'GET Obtener todos' endpoint is highlighted. The request URL is set to `http://localhost:3000/api/pedidos`. The response body is a JSON array containing five pedido objects:

```
[{"id_pedido": 1, "fecha": "2024-04-24T05:00:00.000Z", "id_rest": 1, "total": "00000.00"}, {"id_pedido": 2, "fecha": "2025-01-26T05:00:00.000Z", "id_rest": 78, "total": "18413.00"}, {"id_pedido": 3, "fecha": "2024-06-15T05:00:00.000Z", "id_rest": 41, "total": "62653.00"}, {"id_pedido": 4, "fecha": "2025-03-21T05:00:00.000Z", "id_rest": 42, "total": "27675.00"}, {"id_pedido": 5, "fecha": null, "id_rest": null, "total": null}]
```

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

The screenshot shows the Postman interface with the same setup as the previous one, but the 'GET Obtener por ID' endpoint under the 'Pedido' folder is highlighted. The request URL is set to `http://localhost:3000/api/pedidos/13`. The response body is a JSON object representing a single pedido:

```
{ "id_pedido": 13, "fecha": "2025-03-06T05:00:00.000Z", "id_rest": 48, "total": "86596.00"}
```

Crea un pedido

The screenshot shows the Postman interface with a dark theme. On the left, the 'My Workspace' sidebar lists collections: 'Collections', 'Environments', 'Flows', and 'History'. Under the 'Api' collection, there are several sub-folders: 'Empleado', 'Restaurante', 'Producto', 'Detalle_pedido', and 'Pedido'. The 'Pedido' folder contains methods: 'GET Obtener todos', 'GET Obtener por ID', 'POST Crear', 'PUT Actualizar', 'DEL Eliminar', 'GET Get data', 'PUT Update data', 'POST Post data', and 'DEL Delete data'. The 'POST Crear' method is selected and highlighted with an orange border. The main workspace shows a POST request to 'http://localhost:3000/api/pedidos'. The 'Body' tab is selected, showing raw JSON data:

```
1 {  
2   "fecha": "2025-03-22",  
3   "id_test": 103,  
4   "total": 40000  
5 }
```

Actualiza un pedido

The screenshot shows the Postman interface with a dark theme. The 'My Workspace' sidebar is identical to the previous screenshot. The 'Pedido' folder in the 'Api' collection has its 'PUT Actualizar' method selected. The main workspace shows a PUT request to 'http://localhost:3000/api/pedidos/1'. The 'Headers' tab is selected, showing 'Content-Type: application/json'. The 'Body' tab shows 'Query Params' with a single entry: 'Key' and 'Value' both set to 'Key'.

Elimina un pedido

The screenshot shows the Postman interface with a dark theme. The 'My Workspace' sidebar is identical to the previous screenshots. The 'Pedido' folder in the 'Api' collection has its 'DEL Eliminar' method selected. The main workspace shows a DELETE request to 'http://localhost:3000/api/pedidos/1'. The 'Headers' tab is selected, showing 'Content-Type: application/json'. The 'Body' tab shows 'Query Params' with a single entry: 'Key' and 'Value' both set to 'Key'.

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

The screenshot shows the Postman interface with a red box highlighting the left sidebar where the 'My Workspace' collection is listed. The collection contains several sub-endpoints for different resources like Empleado, Restaurante, Producto, and Detalle_pedido.

Ingresamos 100 registros por tabla

The screenshot shows the pgAdmin 4 interface. The Object Explorer on the left shows a database named 'postres'. The main area has a query editor window with the following SQL script:

```

494 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (86, 47, 35, 2, 22683);
495 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (87, 29, 59, 3, 28356);
496 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (88, 90, 86, 1, 19927);
497 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (89, 62, 57, 4, 14028);
498 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (90, 72, 42, 3, 26528);
499 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (91, 64, 97, 4, 17316);
500 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (92, 58, 62, 5, 14198);
501 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (93, 24, 11, 1, 29986);
502 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (94, 30, 72, 4, 27076);
503 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (95, 66, 47, 2, 28419);
504 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (96, 101, 40, 5, 9833);
505 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (97, 76, 2, 2, 23618);
506 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (98, 15, 24, 5, 14800);
507 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (99, 57, 3, 4, 13944);
508 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (100, 89, 42, 1, 8827);
509 talle, id_pedido, id_prod, cantidad, subtotal) VALUES (101, 50, 24, 1, 8396);

```

Ahora visualizamos los datos de cada tabla

`select * from restaurante;`

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

- Servers (2)
 - Parcia2Bases
 - Databases (1)
 - postgres

Query Query History

```
1 select * from restaurante;
```

Data Output Messages Notifications

	id_rest [PK] integer	nombre text	ciudad text	direccion text	fecha_apertura date
1	1	El Fogón Criollo	Bogotá	Calle 123 #45-...	2023-01-15
2	2	Restaurante 2	Barranquilla	Calle 57 # 15-18	2023-11-08
3	3	Restaurante 3	Medellín	Calle 97 # 42-20	2024-04-28
4	4	Restaurante 4	Barranquilla	Calle 18 # 44-17	2023-01-23
5	5	Restaurante 5	Barranquilla	Calle 26 # 47-13	2024-03-22
6	6	Restaurante 6	Medellín	Calle 90 # 39-9	2024-09-22
7	7	Restaurante 7	Barranquilla	Calle 83 # 23-16	2025-01-01
8	8	Restaurante 8	Bogotá	Calle 93 # 23-1	2023-05-10
9	9	Restaurante 9	Medellín	Calle 74 # 18-9	2024-09-08
10	10	Restaurante 10	Bogotá	Calle 6 # 20-9	2023-02-26
11	11	Restaurante 11	Bogotá	Calle 24 # 46-9	2024-04-29
12	12	Restaurante 12	Bogotá	Calle 51 # 39-3	2024-11-05
13	13	Restaurante 13	Bogotá	Calle 95 # 10-18	2024-02-20
14	14	Restaurante 14	Cali	Calle 42 # 36-1	2023-05-28
15	15	Restaurante 15	Barranquilla	Calle 54 # 41-5	2022-12-03
16	16	Restaurante 16	Bogotá	Calle 97 # 33-3	2024-09-07
17	17	Restaurante 17	Cali	Calle 54 # 36-20	2023-08-13
18	18	Restaurante 18	Barranquilla	Calle 26 # 23-19	2024-10-23
19	19	Restaurante 19	Cali	Calle 23 # 47-8	2023-04-08
20	20	Restaurante 20	Medellín	Calle 24 # 33-16	2024-01-24
21	21	Restaurante 21	Bogotá	Calle 34 # 18-6	2024-10-15
22	22	Restaurante 22	Barranquilla	Calle 88 # 26-17	2024-12-09

Total rows: 101 Query complete 00:00:01.305 CRLF Ln 2, Col 1

select * from producto;

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

- Servers (2)
 - Parcia2Bases
 - Databases (1)
 - postgres

Query Query History

```
1 select * from producto;
```

Data Output Messages Notifications

	id_prod [PK] integer	nombre character varying (100)	precio numeric (10,2)
1	1	Combo Hamburguesa	25000.00
2	2	Producto 2	28493.00
3	3	Producto 3	19887.00
4	4	Producto 4	15978.00
5	5	Producto 5	22869.00
6	6	Producto 6	29364.00
7	7	Producto 7	14596.00
8	8	Producto 8	28425.00
9	9	Producto 9	22285.00
10	10	Producto 10	8420.00
11	11	Producto 11	22280.00
12	12	Producto 12	22532.00
13	13	Producto 13	15458.00
14	14	Producto 14	10909.00
15	15	Producto 15	22194.00
16	16	Producto 16	18228.00
17	17	Producto 17	19669.00
18	18	Producto 18	16285.00
19	19	Producto 19	13922.00
20	20	Producto 20	5015.00
21	21	Producto 21	16430.00
22	22	Producto 22	25691.00

Total rows: 101 Query complete 00:00:01.720 CRLF Ln 2, Col 1

✓ Successfully run. Total query runtime: 1 secs 720 msec. 101 rows affected. X

`select * from empleado;`

The screenshot shows the pgAdmin 4 interface with the 'Object Explorer' on the left and a 'Query Editor' window on the right. The 'Query Editor' displays the following SQL query and its results:

```
1 select * from empleado;
```

The results table has columns: id_empleado [PK] integer, nombre character varying (100), rol character varying (50), and id_nest integer. The data shows 101 rows of employee information.

	id_empleado [PK] integer	nombre character varying (100)	rol character varying (50)	id_nest integer
1	1	Ana Garcia	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 CRLF Ln 1, Col 24

`select * from detalle_pedido;`

The screenshot shows the pgAdmin 4 interface with the 'Object Explorer' on the left and a 'Query Editor' window on the right. The 'Query Editor' displays the following SQL query and its results:

```
1 select * from detalle_pedido;
```

The results table has columns: id_detalle [PK] integer, id_pedido integer, id_prod integer, cantidad integer, and subtotal numeric (10,2). The data shows 101 rows of purchase detail information.

	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	2888.00

Successfully run. Total query runtime: 1 secs 101 msec. 101 rows affected. CRLF Ln 1, Col 1

`select * from pedido;`

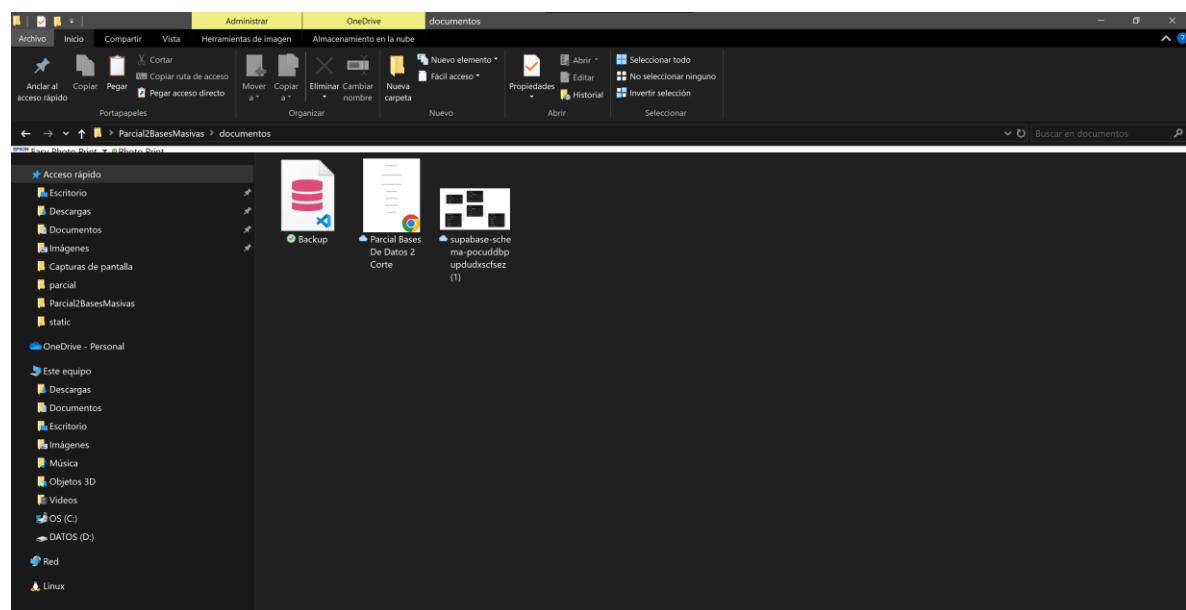
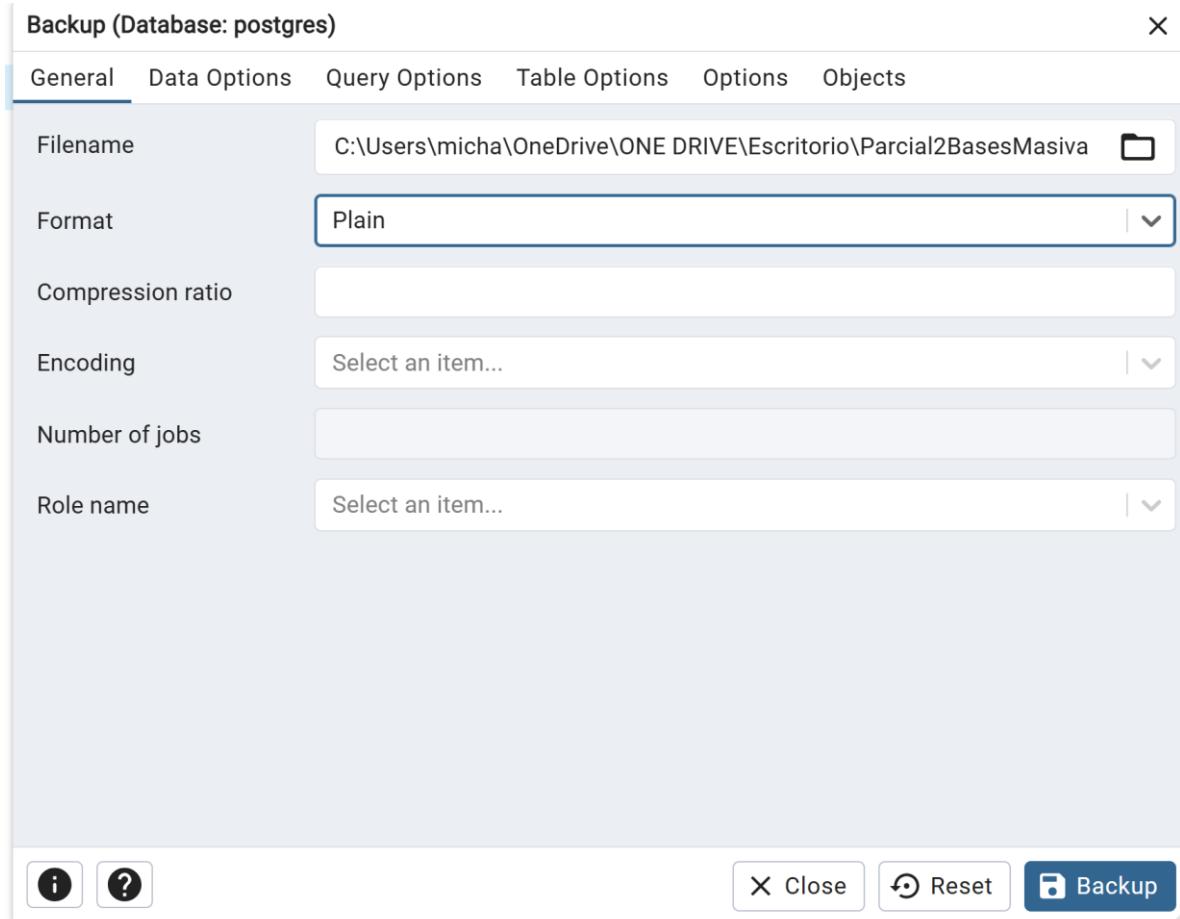
The screenshot shows the pgAdmin 4 interface. On the left is the Object Explorer tree, which is expanded to show the structure of the 'postgres' database under 'Parcia2Bases'. In the main pane, a query window displays the result of the 'select * from pedido;' query. The results are presented in a table with the following columns: id_pedido (PK) integer, fecha date, id_res integer, and total numeric (10,2). The table contains 101 rows of data, showing various dates from 2024-04-24 to 2025-04-18 and total values ranging from 18413.00 to 80193.00.

	id_pedido (PK) integer	fecha date	id_res integer	total numeric (10,2)
1	1	2024-04-24	1	50000.00
2	2	2025-01-26	78	18413.00
3	3	2024-06-15	41	62653.00
4	4	2025-03-21	42	27675.00
5	5	2024-08-16	3	80950.00
6	6	2024-10-21	58	68190.00
7	7	2024-09-07	17	17718.00
8	8	2025-01-21	10	36560.00
9	9	2024-12-28	34	16354.00
10	10	2024-09-27	92	23669.00
11	11	2024-08-28	51	13407.00
12	12	2024-07-01	83	56487.00
13	13	2025-03-06	48	86596.00
14	14	2024-07-23	13	10036.00
15	15	2024-10-31	48	75803.00
16	16	2024-10-15	84	56124.00
17	17	2024-10-06	53	86635.00
18	18	2024-11-22	75	59879.00
19	19	2024-08-23	100	85779.00
20	20	2025-01-05	53	89832.00
21	21	2024-05-02	4	78920.00
22	22	2025-04-18	29	80193.00

También podemos validar en supabase los datos que anteriormente ingresamos

The screenshot shows the Supabase dashboard for the project 'michaelcast01's Project'. The left sidebar has a 'Database' section with 'Tables' selected, showing options like Functions, Triggers, and Extensions. The main area is titled 'Database Tables' and lists the following tables: detalle_pedido, empleado, pedido, producto, and restaurante. Each table entry includes a 'Name', 'Description', 'Rows (Estimated)', 'Size (Estimated)', 'Realtime Enabled' status, and a '5 columns' button.

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

The screenshot shows the Postman application interface. On the left, there's a sidebar titled 'My Workspace' with a tree view of collections: 'Collections' (selected), 'Environments', 'Flows', and 'History'. Under 'Collections', there are several items: 'Api' (selected), 'Empleado', 'Restaurante', 'Producto', 'Detalle_pedido', 'Pedido', and 'Obtener todos'. Each item has associated requests like 'GET Obtener todos', 'POST Crear', etc. The main workspace shows an 'Api' collection with 29 requests, 0 forks, 0 watchers, and was created by 'You' on April 25, 2025. A context menu is open over one of the requests, with 'More' expanded to show 'Step 2:1' and 'Step 3: Export'. The 'Export' option is highlighted.

The screenshot shows a file manager interface with a dark theme. It displays a list of files under a folder named 'Hoy (2)'. There are two entries: 'Api.postman_collection' (modified on 4/25/2025 at 12:05 PM) and 'Backup' (modified on 4/25/2025 at 11:58 AM). Both files are listed as 'Archivo de origen ...' with their respective file sizes: 19 KB and 219 KB.

The screenshot shows the OneDrive interface with a dark theme. It displays a file list under a folder named 'Parcial2BasesMasivas > documentos'. The 'Backup' file is selected, indicated by a checkmark icon. Other files visible include 'Api.postman_collection' and 'Parcial Bases De Datos 2 Corte'. The OneDrive ribbon at the top includes tabs for 'Archivo', 'Inicio', 'Compartir', 'Vista', and 'Almacenamiento en la nube'.