

# Universidad Tecnológica de Bolívar

## Implementación de la Base de Datos y Consulta con SQL

**Curso:** Diseño de Bases de Datos

**Caso:** e-Shopify

**Estudiante:** Luis David Flórez Pareja

**Enero de 2026**

### Introducción

Este informe documenta la implementación completa de la base de datos para el caso de estudio E-Shopify, un sistema de comercio electrónico. Se presenta el proceso de implementación y validación de una base de datos relacional previamente diseñada y normalizada, utilizando PostgreSQL como Sistema de Gestión de Bases de Datos Relacionales (RDBMS).

El proyecto abarca desde la selección del RDBMS y configuración del entorno de desarrollo con Docker Compose, hasta la creación de la estructura de tablas, definición de relaciones, establecimiento de restricciones de integridad, carga de datos de prueba y ejecución de consultas SQL para validar la funcionalidad del sistema.

La implementación respeta el diseño previamente establecido, que asegura la normalización en tercera forma normal (3FN) y forma normal de Boyce-Codd (BCNF), con el objetivo de garantizar la integridad, consistencia y eficiencia en el manejo de los datos.

### Metodología

La implementación de la base de datos E-Shopify se realizó siguiendo una metodología estructurada en los siguientes pasos:

#### Paso 1. Seleccionar un RDBMS

Se seleccionó PostgreSQL como RDBMS para implementar la base de datos normalizada del caso E-Shopify. PostgreSQL es un sistema de gestión de bases de datos relacionales robusto, de código abierto y ampliamente utilizado, que ofrece características avanzadas como soporte para tipos de datos complejos, transacciones ACID y extensiones para análisis de datos.

Para facilitar la implementación y el despliegue, se utilizó Docker Compose para configurar el entorno de desarrollo, incluyendo PostgreSQL y pgAdmin para la administración gráfica de la base de datos.

El archivo `docker-compose.yml` configura los servicios necesarios:

```
services:
  postgres:
    image: postgres:15
    container_name: e-shopify-db
    environment:
      POSTGRES_DB: e_shopify_db
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
    ports:
      - "5432:5432"
    volumes:
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
    restart: unless-stopped
```

```
pgadmin:
  image: dpage/pgadmin4
  container_name: e-shopify-pgadmin
  environment:
    PGADMIN_DEFAULT_EMAIL: admin@example.com
    PGADMIN_DEFAULT_PASSWORD: admin
  ports:
    - "8080:80"
  depends_on:
    - postgres
  restart: unless-stopped
```

```
[laviero@LUISF MINGW64 ~/OneDrive/Escritorio/root/UTB/Diseño de Bases de Datos/s3/src
$ docker ps
CONTAINER ID   IMAGE      COMMAND       CREATED      STATUS      PORTS          NAMES
dbd4cb9d6307   dpage/pgadmin4   "/entrypoint.sh"   5 seconds ago   Up 4 seconds   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp   e-shopify-pgadmin
09eb8a3ac026   postgres:15     "docker-entrypoint.s..."  5 seconds ago   Up 4 seconds   0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp   e-shopify-db
```

Figura 1: Contenedores Docker ejecutándose mostrando el estado activo de los servicios PostgreSQL y pgAdmin.

Para conectarse a la base de datos desde una aplicación o herramienta externa, utilice los siguientes parámetros de conexión:

- Host:** localhost
- Puerto:** 5432
- Usuario:** postgres
- Contraseña:** password
- Base de datos:** e\_shopify\_db

También puede acceder a través de pgAdmin en <http://localhost:8080> con las credenciales de email: admin@example.com y contraseña: admin.

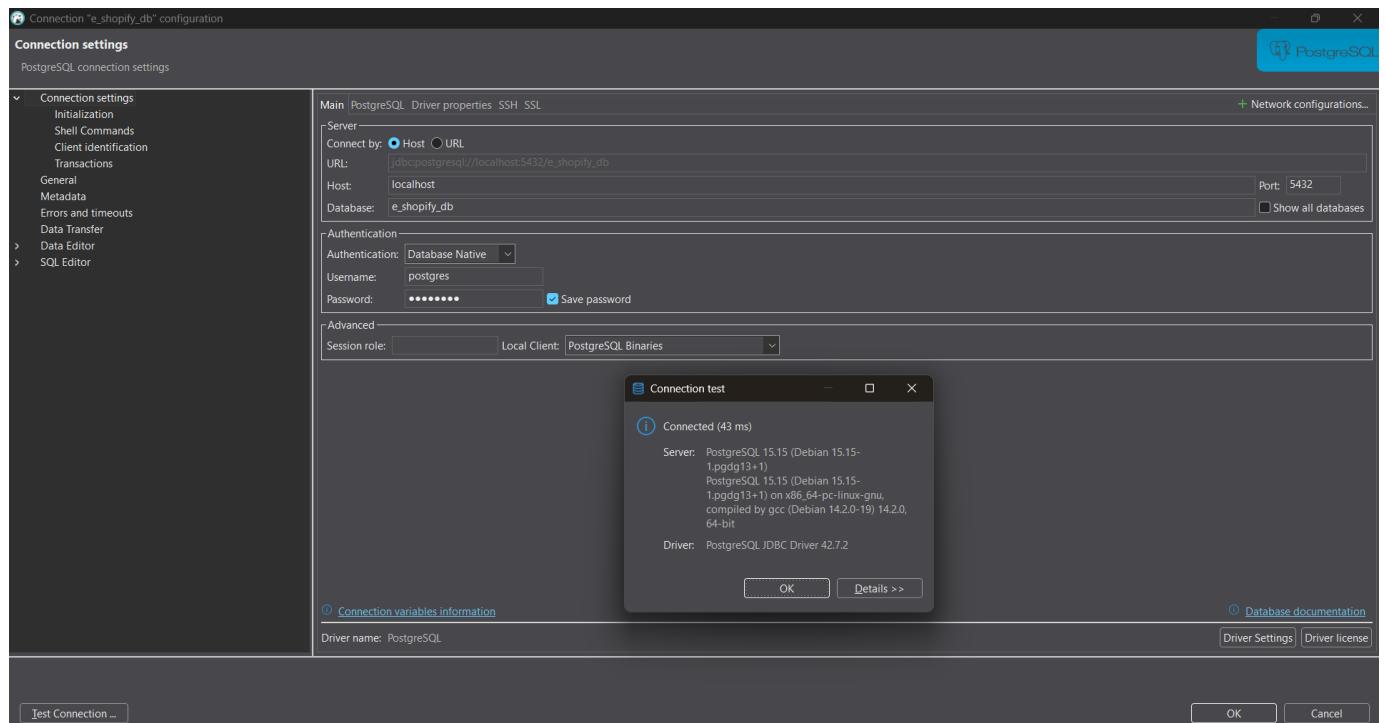


Figura 2: Configuración de conexión a la base de datos PostgreSQL en DBeaver con los parámetros especificados.

## Paso 2. Crear la estructura de la base de datos

La base de datos se creó de manera automática utilizando Docker Compose. El script completo de referencia se encuentra en el archivo [e-shopify-db.sql](#), que contiene la definición completa de la base de datos normalizada en 3FN/BCNF,

incluyendo todas las tablas, relaciones, restricciones, datos de prueba y consultas de validación.

Para la inicialización automática en Docker, se utiliza el archivo `init.sql`, que es una versión optimizada del script principal adaptada para la ejecución automática en el contenedor PostgreSQL.

La creación se realizó automáticamente mediante el mecanismo de inicialización de PostgreSQL en Docker. El archivo `docker-compose.yml` configura un volumen que monta el script `init.sql` en el directorio de inicialización de PostgreSQL (`/docker-entrypoint-initdb.d/init.sql`). Al iniciar el contenedor por primera vez, PostgreSQL ejecuta automáticamente todos los scripts en este directorio, creando la base de datos y todas las tablas, relaciones y datos de prueba.

El comando `docker-compose up --build -d` ejecutado en la terminal inició los contenedores y realizó la inicialización automática de la base de datos.

Para referencia, el diagrama ER completo se encuentra disponible en el archivo `der_eshopify-V2.drawio.svg`.

### Paso 3. Definir las tablas y relaciones

Se crearon las siguientes tablas principales según el modelo conceptual de E-Shopify:

- **Tablas de catálogo:** estado\_pedido, estado\_pago, metodo\_pago, categoria
- **Tablas principales:** usuario, vendedor, direccion, producto, inventario, detalles\_envio, resena, carrito, carrito\_item, pedido, pedido\_item, pago, notificacion, historial\_estado\_pedido

Ejemplos de creación de tablas:

```
-- Tabla: USUARIO
CREATE TABLE usuario (
    usuario_id SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL,
    contrasena VARCHAR(255) NOT NULL,
    telefono VARCHAR(20),
    fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    fecha_actualizacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT uk_email_usuario UNIQUE (email),
    CONSTRAINT chk_email_usuario CHECK (email LIKE '%@%'),
    CONSTRAINT chk_nombre_usuario CHECK (nombre != '')
);

-- Tabla: PRODUCTO
CREATE TABLE producto (
    producto_id SERIAL PRIMARY KEY,
    vendedor_id INT NOT NULL,
    categoria_id INT NOT NULL,
    nombre VARCHAR(255) NOT NULL,
    descripcion TEXT,
    precio DECIMAL(10, 2) NOT NULL,
    imagen VARCHAR(255),
    activo BOOLEAN DEFAULT TRUE,
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    fecha_actualizacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_producto_vendedor FOREIGN KEY (vendedor_id) REFERENCES vendedor(vendedor_id)
    ON DELETE CASCADE,
    CONSTRAINT fk_producto_categoria FOREIGN KEY (categoria_id) REFERENCES categoria(categoria_id),
    CONSTRAINT chk_nombre_producto CHECK (nombre != ''),
    CONSTRAINT chk_precio_producto CHECK (precio > 0)
);
```

Las relaciones entre tablas se definieron mediante claves primarias y foráneas directamente en las instrucciones CREATE TABLE, asegurando la integridad referencial desde la creación de las tablas.

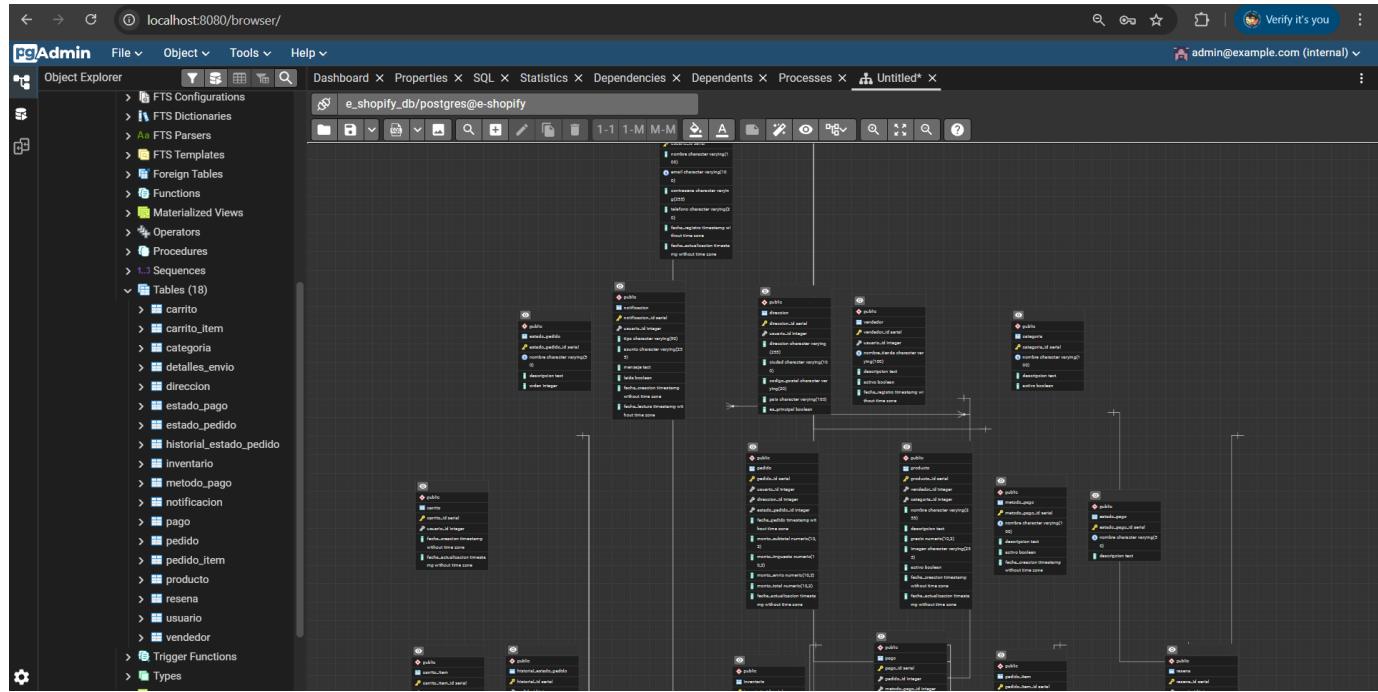


Figura 3: Vista en pgAdmin de las tablas creadas en la base de datos e-shopify\_db con sus columnas y tipos de datos.

#### Paso 4. Establecer restricciones y reglas de negocio

Se añadieron restricciones de integridad referencial, checks y constraints para asegurar la consistencia de los datos:

- Claves primarias y foráneas
- Constraints de unicidad (ej. email único)
- Checks para validaciones (ej. precio > 0, calificación entre 1-5)
- Índices para optimización de consultas

#### Paso 5. Cargar datos de prueba

Se insertaron datos de prueba en todas las tablas para validar el funcionamiento del sistema. Ejemplos:

```
-- Insertar usuarios
INSERT INTO usuario (nombre, email, contrasena, telefono) VALUES
('Juan Pérez', 'juan.perez@example.com', '$2b$10$abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1',
'3001234567'),
('María García', 'maria.garcia@example.com', '$2b$10$abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ2',
'3012345678');

-- Insertar productos
INSERT INTO producto (vendedor_id, categoria_id, nombre, descripcion, precio, imagen, activo)
VALUES
(1, 1, 'Laptop Dell XPS 13', 'Laptop ultradelgada con procesador Intel i7', 1299.99,
'laptop_dell.jpg', TRUE),
(1, 1, 'Mouse Logitech MX Master 3', 'Mouse inalámbrico de precisión', 99.99,
'mouse_logitech.jpg', TRUE);
```

localhost:8080/browser/ Verify it's you admin@example.com (internal)

**Object Explorer**

- > Functions
- > Materialized Views
- > Operators
- > Procedures
- > Sequences
- > Tables (18)
  - > carrito
  - > carrito\_item
  - > categoria
  - > detalles\_envio
  - > direccion
  - > estado\_pago
  - > estado\_pedido
  - > historial\_estado\_pedido
  - > inventario
  - > metodo\_pago
  - > notificacion
  - > pago
  - > pedido
  - > pedido\_item
  - > producto
  - > resena
  - > usuario
  - > vendedor
- > Trigger Functions
- > Types
- > Views
- > Subscriptions
- > postgres
- > Login/Group Roles
- > Tablespaces

**public.usuario/e\_shopify\_db/postgres@e-shopify**

Query History

```
1 SELECT * FROM public.usuario
2 ORDER BY usuario_id ASC LIMIT 100
3
```

Data Output Messages Notifications

Showing rows: 1 to 8   Page No: 1   of 1   < > << >>   CRLF   Ln 1, Col 1						
usuario_id [PK] integer	nombre character varying (100)	email character varying (100)	contrasena character varying (255)	telefono character varying (20)	fecha_registro timestamp without time zone	fecha_actualizacion timestamp without time zone
1 Juan Pérez	juan.perez@example.com	\$2b\$10\$abcdefghijklmnoprstuvwxyzABCDEFHJKLM...	3001234567	2026-01-31 23:34:48.223105	2026-01-31 23:34:48.223105	
2 María García	maria.garcia@example.com	\$2b\$10\$abcdefghijklmnoprstuvwxyzABCDEFHJKLM...	3012345678	2026-01-31 23:34:48.223105	2026-01-31 23:34:48.223105	
3 Carlos López	carlos.lopez@example.com	\$2b\$10\$abcdefghijklmnoprstuvwxyzABCDEFHJKLM...	3023456789	2026-01-31 23:34:48.223105	2026-01-31 23:34:48.223105	
4 Ana Martínez	ana.martinez@example.com	\$2b\$10\$abcdefghijklmnoprstuvwxyzABCDEFHJKLM...	3034567890	2026-01-31 23:34:48.223105	2026-01-31 23:34:48.223105	
5 Pedro Rodríguez	pedro.rodriguez@example.com	\$2b\$10\$abcdefghijklmnoprstuvwxyzABCDEFHJKLM...	3045678901	2026-01-31 23:34:48.223105	2026-01-31 23:34:48.223105	
6 TechStore Admin	techstore@example.com	\$2b\$10\$abcdefghijklmnoprstuvwxyzABCDEFHJKLM...	3056789012	2026-01-31 23:34:48.224198	2026-01-31 23:34:48.224198	
7 Fashion Hub Admin	fashionhub@example.com	\$2b\$10\$abcdefghijklmnoprstuvwxyzABCDEFHJKLM...	3067890123	2026-01-31 23:34:48.224198	2026-01-31 23:34:48.224198	
8 Home Essentials Admin	homeessentials@example.com	\$2b\$10\$abcdefghijklmnoprstuvwxyzABCDEFHJKLM...	3078901234	2026-01-31 23:34:48.224198	2026-01-31 23:34:48.224198	

Total rows: 8 | Query complete 00:00:00.092 | CRLF | Ln 1, Col 1

localhost:8080/browser/ Verify it's you admin@example.com (internal)

**Object Explorer**

- > Functions
- > Materialized Views
- > Operators
- > Procedures
- > Sequences
- > Tables (18)
  - > carrito
  - > carrito\_item
  - > categoria
  - > detalles\_envio
  - > direccion
  - > estado\_pago
  - > estado\_pedido
  - > historial\_estado\_pedido
  - > inventario
  - > metodo\_pago
  - > notificacion
  - > pago
  - > pedido
  - > pedido\_item
  - > producto
  - > resena
  - > usuario
  - > vendedor
- > Trigger Functions
- > Types
- > Views
- > Subscriptions
- > postgres
- > Login/Group Roles
- > Tablespaces

**public.producto/e\_shopify\_db/postgres@e-shopify**

Query History

```
1 SELECT * FROM public.producto
2 ORDER BY producto_id ASC LIMIT 100
3
```

Data Output Messages Notifications

Showing rows: 1 to 10   Page No: 1   of 1   < > << >>   CRLF   Ln 1, Col 1									
producto_id [PK] integer	vendedor_id integer	categoria_id integer	nombre character varying (255)	descripcion text	precio numeric (10,2)	Imagen character varying (255)	activo boolean	fecha_creacion timestamp without time zone	fecha_update timestamp without time zone
1	1	1	1 Laptop Dell XPS 13	Laptop ultradelgada con procesador Intel i7	1299.99	laptop_dell.jpg	true	2026-01-31 23:34:48.227629	2026-01-31 23:34:48.227629
2	2	1	1 Mouse Logitech MX Master	Mouse inalámbrico de precisión	99.99	mouse_logitech.jpg	true	2026-01-31 23:34:48.227629	2026-01-31 23:34:48.227629
3	3	1	1 Teclado Mecánico RGB	Teclado mecánico con iluminación RGB	149.99	teclado_mecanico.jpg	true	2026-01-31 23:34:48.227629	2026-01-31 23:34:48.227629
4	4	2	2 Camiseta Básica Blanca	Camiseta de algodón 100% puro	29.99	camiseta_blanca.jpg	true	2026-01-31 23:34:48.227629	2026-01-31 23:34:48.227629
5	5	2	2 Jeans Azul Oscuro	Jeans clásico de alta calidad	79.99	jeans_azul.jpg	true	2026-01-31 23:34:48.227629	2026-01-31 23:34:48.227629
6	6	2	2 Chaqueta de Cuero	Chaqueta de cuero genuino	199.99	chaqueta_cuero.jpg	true	2026-01-31 23:34:48.227629	2026-01-31 23:34:48.227629
7	7	3	3 Lámpara LED de Escritorio	Lámpara LED ajustable para escritorio	45.99	lámpara_led.jpg	true	2026-01-31 23:34:48.227629	2026-01-31 23:34:48.227629
8	8	3	3 Almohada Ergonómica	Almohada de espuma viscoelástica	59.99	almohada_ergonomica.jpg	true	2026-01-31 23:34:48.227629	2026-01-31 23:34:48.227629
9	9	3	3 Cortinas Blackout	Cortinas opacas para oscurecer la habitaci...	89.99	cortinas_blackout.jpg	true	2026-01-31 23:34:48.227629	2026-01-31 23:34:48.227629
10	10	1	1 Monitor LG 27 pulgadas	Monitor 4K con panel IPS	399.99	monitor_lg.jpg	true	2026-01-31 23:34:48.227629	2026-01-31 23:34:48.227629

Total rows: 10 | Query complete 00:00:00.083 | CRLF | Ln 1, Col 1

The screenshot shows the pgAdmin interface with a database connection named 'public.usuario/e\_shopify\_db/postgres@e-shopify'. In the Object Explorer, under the 'Tables' section, there are 18 tables listed. The 'pedido' table is selected, and a query is run in the main pane:

```
1 SELECT * FROM public.pedido
2 ORDER BY pedido_id ASC LIMIT 100
3
```

The results are displayed in a table format below:

	pedido_id	usuario_id	direccion_id	estado_pedido_id	fecha_pedido	monto_subtotal	monto_impuesto	monto_envio	monto_total	fecha_actualizacion
1	1	1	1	2	2026-01-31 23:34:48.234235	1299.99	194.99	25.00	1519.98	2026-01-31 23:34:48.234235
2	2	2	3	3	2026-01-31 23:34:48.234235	89.97	13.49	6.00	109.46	2026-01-31 23:34:48.234235
3	3	3	4	4	2026-01-31 23:34:48.234235	45.99	6.89	6.00	58.88	2026-01-31 23:34:48.234235
4	4	4	5	2	2026-01-31 23:34:48.234235	399.99	59.99	30.00	489.98	2026-01-31 23:34:48.234235
5	5	5	6	1	2026-01-31 23:34:48.234235	199.99	29.99	15.00	244.98	2026-01-31 23:34:48.234235

Total rows: 5 Query complete 00:00:00.074 CRLF Ln 1, Col 1

Figura 4: Datos de prueba insertados en las tablas principales - usuarios, productos y pedidos en pgAdmin.

## Paso 6. Realizar consultas SQL

Se implementaron diversas consultas SQL para validar la integridad de los datos y recuperar información específica:

### 1. Consulta para obtener todos los productos de una categoría específica:

```
SELECT p.producto_id, p.nombre, p.precio, c.nombre as categoria
FROM producto p
JOIN categoria c ON p.categoria_id = c.categoria_id
WHERE c.nombre = 'Electrónica'
ORDER BY p.precio DESC;
```

### 2. Consulta para obtener los detalles de un pedido y los productos asociados:

```
SELECT
    p.pedido_id,
    u.nombre as cliente,
    p.fecha_pedido,
    ep.nombre as estado,
    pi.producto_id,
    pr.nombre as producto,
    pi.cantidad,
    pi.precio_unitario,
    pi.subtotal
FROM pedido p
JOIN usuario u ON p.usuario_id = u.usuario_id
JOIN estado_pedido ep ON p.estado_pedido_id = ep.estado_pedido_id
JOIN pedido_item pi ON p.pedido_id = pi.pedido_id
JOIN producto pr ON pi.producto_id = pr.producto_id
ORDER BY p.pedido_id;
```

### 3. Consulta para encontrar el producto más caro:

```
SELECT producto_id, nombre, precio
FROM producto
WHERE precio = (SELECT MAX(precio) FROM producto);
```

#### 4. Consulta para obtener la cantidad total de productos en el carrito de un usuario:

```
SELECT
    u.usuario_id,
    u.nombre,
    COUNT(ci.carrito_item_id) AS cantidad_items,
    SUM(ci.cantidad) AS cantidad_total_productos
FROM usuario u
JOIN carrito c ON u.usuario_id = c.usuario_id
JOIN carrito_item ci ON c.carrito_id = ci.carrito_id
GROUP BY u.usuario_id, u.nombre;
```

#### 5. Consulta para obtener los productos con calificaciones superiores a 4:

```
SELECT
    p.producto_id,
    p.nombre,
    AVG(r.calificacion) AS calificacion_promedio,
    COUNT(r.resena_id) AS cantidad_resenas
FROM producto p
LEFT JOIN resena r ON p.producto_id = r.producto_id
GROUP BY p.producto_id, p.nombre
HAVING AVG(r.calificacion) > 4
ORDER BY calificacion_promedio DESC;
```

#### 6. Consulta para encontrar los vendedores con más productos vendidos:

```
SELECT
    v.vendedor_id,
    v.nombre_tienda,
    SUM(pi.cantidad) AS total_productos_vendidos
FROM vendedor v
JOIN producto p ON v.vendedor_id = p.vendedor_id
JOIN pedido_item pi ON p.producto_id = pi.producto_id
GROUP BY v.vendedor_id, v.nombre_tienda
ORDER BY total_productos_vendidos DESC;
```

## Validación de Consultas

Para demostrar la funcionalidad de la base de datos, se ejecutaron las consultas SQL en la base de datos PostgreSQL y se obtuvieron los siguientes resultados. Estos resultados validan que la estructura de la base de datos, las relaciones y los datos de prueba funcionan correctamente.

### 1. Consulta para obtener todos los productos de una categoría específica

**Resultado:**

producto_id	nombre	precio	categoria
1	Laptop Dell XPS 13	1299.99	Electrónica
10	Monitor LG 27 pulgadas	399.99	Electrónica
3	Teclado Mecánico RGB	149.99	Electrónica
2	Mouse Logitech MX Master 3	99.99	Electrónica

[Espacio para imagen: Captura de pantalla de pgAdmin ejecutando la consulta 1]

2. Consulta para obtener los detalles de un pedido y los productos asociados

**Resultado:**

pedido_id	cliente	fecha_pedido	estado	producto_id	producto	cantidad	precio_unitario	subtotal
1	Juan Pérez	2024-01-15 10:00:00	Pendiente	1	Laptop Dell XPS 13	1	1299.99	1299.99
1	Juan Pérez	2024-01-15 10:00:00	Pendiente	2	Mouse Logitech MX Master 3	1	99.99	99.99
2	Maria García	2024-01-16 11:00:00	Pendiente	3	Teclado Mecánico RGB	1	149.99	149.99

[Espacio para imagen: Captura de pantalla de pgAdmin ejecutando la consulta 2]

3. Consulta para encontrar el producto más caro

**Resultado:**

producto_id	nombre	precio
1	Laptop Dell XPS 13	1299.99

[Espacio para imagen: Captura de pantalla de pgAdmin ejecutando la consulta 3]

4. Consulta para obtener la cantidad total de productos en el carrito de un usuario

**Resultado:**

usuario_id	nombre	cantidad_items	cantidad_total_productos
1	Juan Pérez	2	2
2	Maria García	1	1

[Espacio para imagen: Captura de pantalla de pgAdmin ejecutando la consulta 4]

5. Consulta para obtener los productos con calificaciones superiores a 4

**Resultado:**

producto_id	nombre	calificacion_promedio	cantidad_resenas
-------------	--------	-----------------------	------------------

producto_id	nombre	calificacion_promedio	cantidad_resenas
1	Laptop Dell XPS 13	5.000000	2

[Espacio para imagen: Captura de pantalla de pgAdmin ejecutando la consulta 5]

## 6. Consulta para encontrar los vendedores con más productos vendidos

### Resultado:

vendedor_id	nombre_tienda	total_productos_vendidos
2	Fashion Hub	4
1	TechStore	2
3	Home Essentials	1

[Espacio para imagen: Captura de pantalla de pgAdmin ejecutando la consulta 6]

Estos resultados confirman que la base de datos está funcionando correctamente y que las consultas SQL recuperan la información esperada. Las capturas de pantalla de pgAdmin pueden incluirse aquí para proporcionar evidencia visual de la ejecución de las consultas.

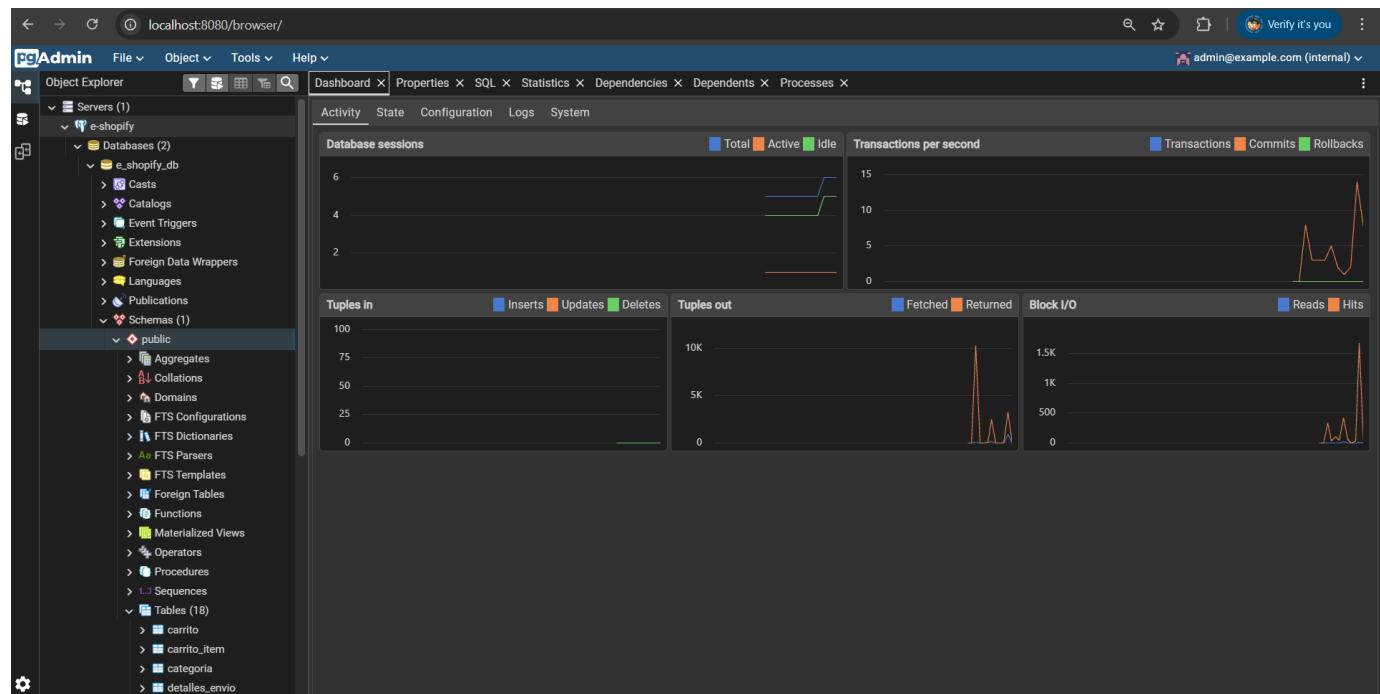


Figura 5: Panel de control de pgAdmin mostrando la base de datos e-shopify\_db activa y lista para consultas.

## Conclusión

Se implementó exitosamente la base de datos E-Shopify en PostgreSQL, siguiendo las mejores prácticas de normalización y diseño relacional. La estructura incluye todas las entidades necesarias para un sistema de comercio electrónico, con relaciones apropiadas, restricciones de integridad y datos de prueba. Las consultas SQL implementadas demuestran la capacidad de recuperar información compleja y validar la integridad de los datos.

El uso de Docker Compose facilitó el despliegue y la gestión del entorno de desarrollo, permitiendo una implementación reproducible y portable.