LUIS FELIPE DIAZ GUTIERREZ

UF1472- LENGUAJE SQL

Documentación: Proyecto SQL-POSTGRES-Northwind

Objetivo:

Implementar consultas SQL avanzadas, vistas, funciones y triggers en la base de datos Northwind (PostgreSQL) para extraer información valiosa, automatizar procesos y mejorar el rendimiento.

1. Configuración Inicial.

1.1. Verificar la Base de Datos Northwind, antes de comenzar, asegurarse de que la base de datos está instalada y accesible:

```
postgres=# \c northwind
Ahora está conectado a la base de datos «northwind» con el usuario «postgres».
northwind=# \d
                 Listado de relaciones
 Esquema |          Nombre           | Tipo  |  Due±o
---------+---------------------------+-------+----------
 public  | categories                | tabla | postgres
 public  | customer_customer_demo    | tabla | postgres
 public  | customer_demographics     | tabla | postgres
 public  | customers                 | tabla | postgres
 public  | employee_territories      | tabla | postgres
 public  | employees                 | tabla | postgres
 public  | order_details             | tabla | postgres
 public  | orders                    | tabla | postgres
 public  | products                  | tabla | postgres
 public  | region                    | tabla | postgres
 public  | shippers                  | tabla | postgres
 public  | suppliers                 | tabla | postgres
 public  | territories               | tabla | postgres
 public  | us_states                 | tabla | postgres
 public  | vw_analisis_clientes      | vista | postgres
 public  | vw_ordenes_ciudad         | vista | postgres
 public  | vw_precios_categoria      | vista | postgres
 public  | vw_productos_mas_vendidos | vista | postgres
 public  | vw_stock_bajo             | vista | postgres
 public  | vw_top_clientes           | vista | postgres
 public  | vw_top_productos          | vista | postgres
 public  | vw_ventas_categori        | vista | postgres
 public  | vw_ventas_diarias         | vista | postgres
 public  | vw_ventas_empleado        | vista | postgres
 public  | vw_ventas_mensuales       | vista | postgres
 public  | vw_ventas_mes             | vista | postgres
-- Más  --
```

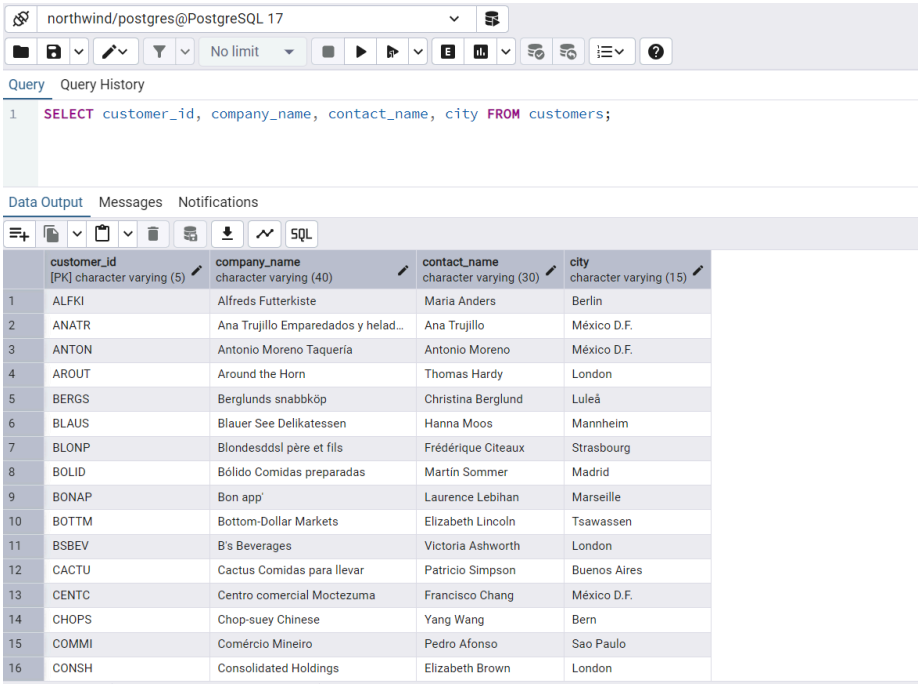
2. Consultas SQL Básicas.

2.1. Consultas de Selección.

Ejemplos esenciales para familiarizarse con los datos:

- Listar todos los clientes:

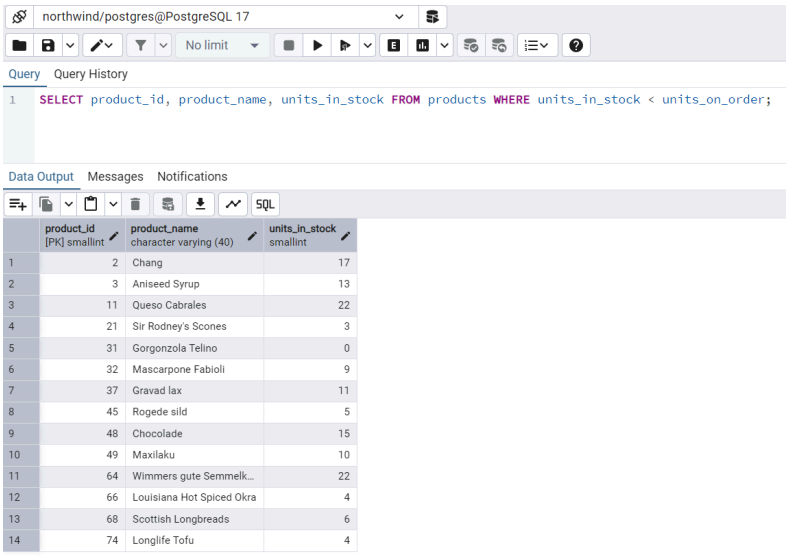
SELECT customer_id, company_name, contact_name, city FROM customers;



	customer_id [PK] character varying (5)	company_name character varying (40)	contact_name character varying (30)	city character varying (15)
1	ALFKI	Alfreds Futterkiste	Maria Anders	Berlin
2	ANATR	Ana Trujillo Emparedados y helad...	Ana Trujillo	México D.F.
3	ANTON	Antonio Moreno Taquería	Antonio Moreno	México D.F.
4	AROUT	Around the Horn	Thomas Hardy	London
5	BERGS	Berglunds snabbköp	Christina Berglund	Luleå
6	BLAUS	Blauer See Delikatessen	Hanna Moos	Mannheim
7	BLONP	Blondesddsl père et fils	Frédérique Citeaux	Strasbourg
8	BOLID	Bólido Comidas preparadas	Martín Sommer	Madrid
9	BONAP	Bon app'	Laurence Lebihan	Marseille
10	BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Tsawassen
11	BSBEV	B's Beverages	Victoria Ashworth	London
12	CACTU	Cactus Comidas para llevar	Patricio Simpson	Buenos Aires
13	CENTC	Centro comercial Moctezuma	Francisco Chang	México D.F.
14	CHOPS	Chop-suey Chinese	Yang Wang	Bern
15	COMMI	Comércio Mineiro	Pedro Afonso	Sao Paulo
16	CONSH	Consolidated Holdings	Elizabeth Brown	London

- Productos con stock bajo

SELECT product_id, product_name, units_in_stock FROM products WHERE units_in_stock < units_on_order;

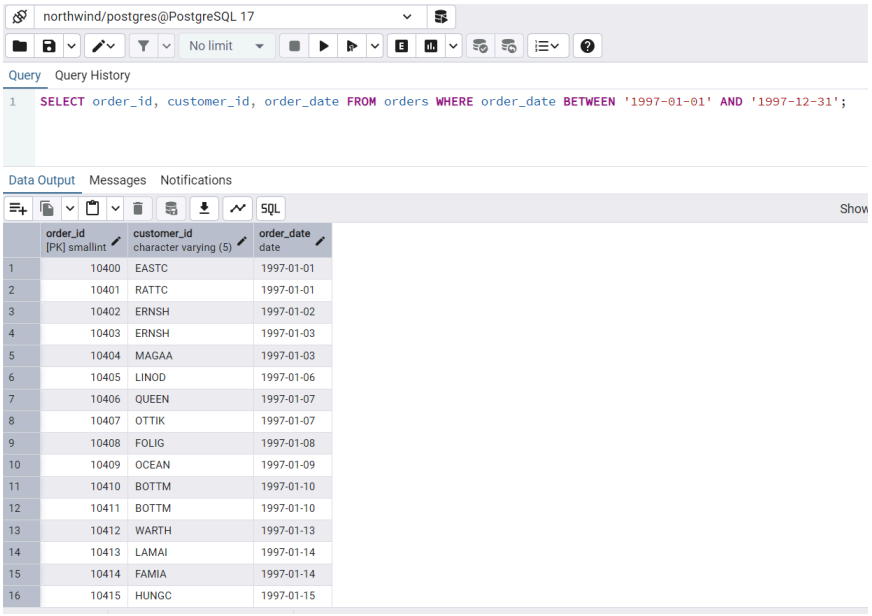


	product_id [PK] smallint	product_name character varying (40)	units_in_stock smallint
1	2	Chang	17
2	3	Aniseed Syrup	13
3	11	Queso Cabrales	22
4	21	Sir Rodney's Scones	3
5	31	Gorgonzola Telino	0
6	32	Mascarpone Fabioli	9
7	37	Gravad lax	11
8	45	Rogede sild	5
9	48	Chocolade	15
10	49	Maxilaku	10
11	64	Wimmers gute Semmelk...	22
12	66	Louisiana Hot Spiced Okra	4
13	68	Scottish Longbreads	6
14	74	Longlife Tofu	4

2.2. Consultas con Filtros y Ordenación.

- Pedidos realizados en 1997:

  SELECT order_id, customer_id, order_date FROM orders WHERE order_date BETWEEN '1997-01-01' AND '1997-12-31';

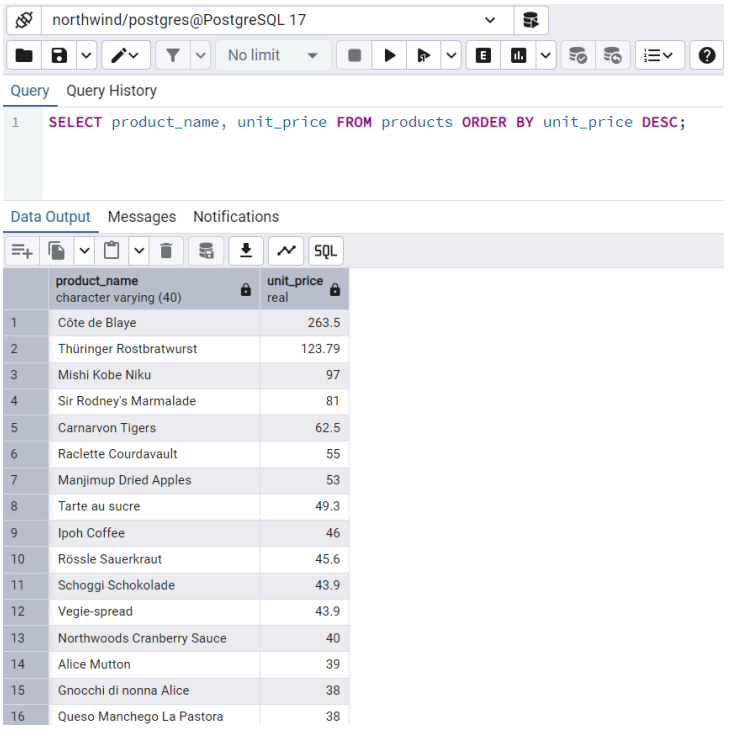


```
SELECT order_id, customer_id, order_date FROM orders WHERE order_date BETWEEN '1997-01-01' AND '1997-12-31';
```

	order_id [PK] smallint	customer_id character varying (5)	order_date date
1	10400	EASTC	1997-01-01
2	10401	RATTC	1997-01-01
3	10402	ERNSH	1997-01-02
4	10403	ERNSH	1997-01-03
5	10404	MAGAA	1997-01-03
6	10405	LINOD	1997-01-06
7	10406	QUEEN	1997-01-07
8	10407	OTTIK	1997-01-07
9	10408	FOLIG	1997-01-08
10	10409	OCEAN	1997-01-09
11	10410	BOTTM	1997-01-10
12	10411	BOTTM	1997-01-10
13	10412	WARTH	1997-01-13
14	10413	LAMAI	1997-01-14
15	10414	FAMIA	1997-01-14
16	10415	HUNGC	1997-01-15

- Productos ordenados por precio descendente

  SELECT product_name, unit_price FROM products ORDER BY unit_price DESC;



```
SELECT product_name, unit_price FROM products ORDER BY unit_price DESC;
```

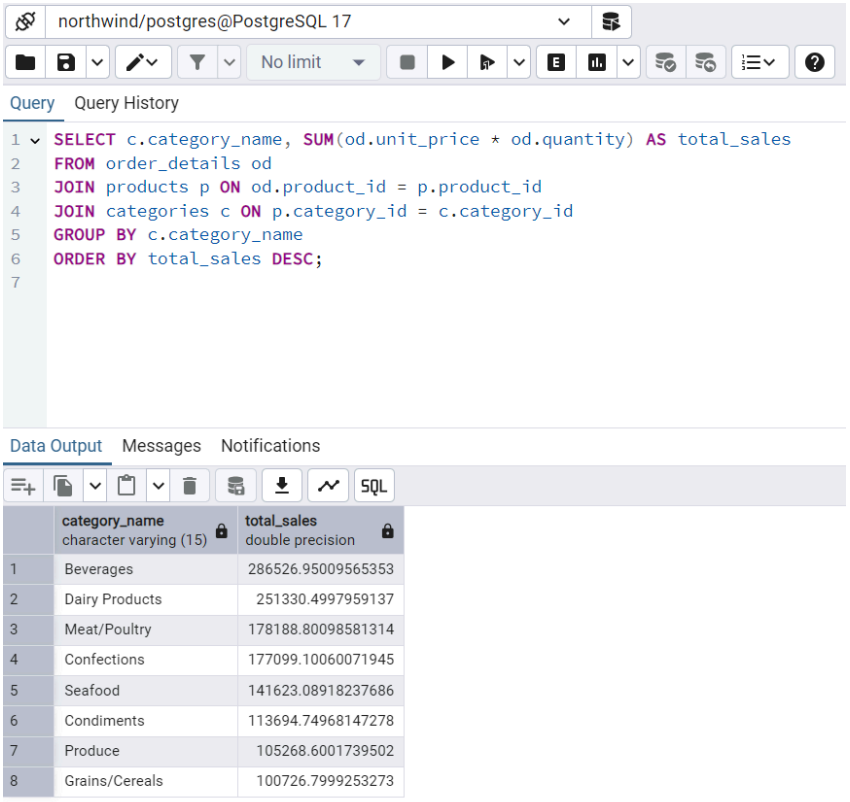
	product_name character varying (40)	unit_price real
1	Côte de Blaye	263.5
2	Thüringer Rostbratwurst	123.79
3	Mishi Kobe Niku	97
4	Sir Rodney's Marmalade	81
5	Carnarvon Tigers	62.5
6	Raclette Courdavault	55
7	Manjimup Dried Apples	53
8	Tarte au sucre	49.3
9	Ipoh Coffee	46
10	Rössle Sauerkraut	45.6
11	Schoggi Schokolade	43.9
12	Vegie-spread	43.9
13	Northwoods Cranberry Sauce	40
14	Alice Mutton	39
15	Gnocchi di nonna Alice	38
16	Queso Manchego La Pastora	38

3. Consultas Avanzadas.

3.1. Joins y Agregaciones.

- **Ventas totales por categoría:**

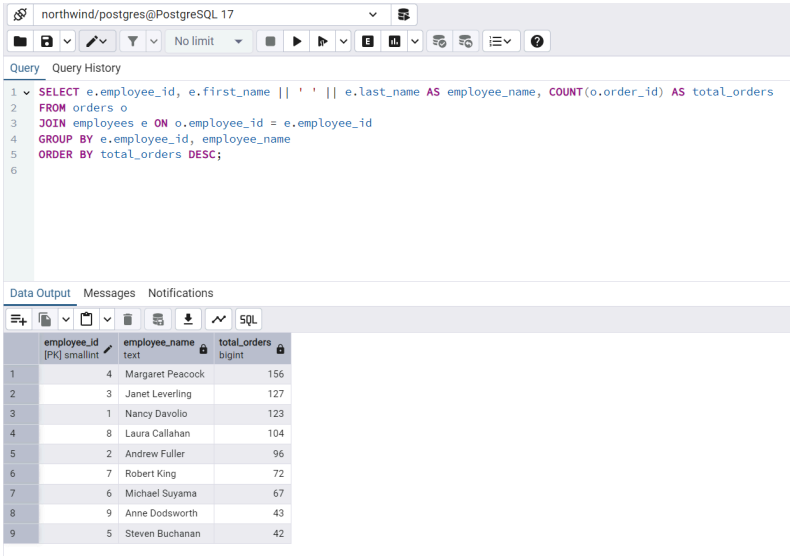
```
SELECT c.category_name, SUM(od.unit_price * od.quantity) AS total_sales
FROM order_details od
JOIN products p ON od.product_id = p.product_id
JOIN categories c ON p.category_id = c.category_id
GROUP BY c.category_name
ORDER BY total_sales DESC;
```



	category_name character varying (15)	total_sales double precision
1	Beverages	286526.95009565353
2	Dairy Products	251330.4997959137
3	Meat/Poultry	178188.80098581314
4	Confections	177099.10060071945
5	Seafood	141623.08918237686
6	Condiments	113694.74968147278
7	Produce	105268.6001739502
8	Grains/Cereals	100726.7999253273

- **Empleados con más ventas:**

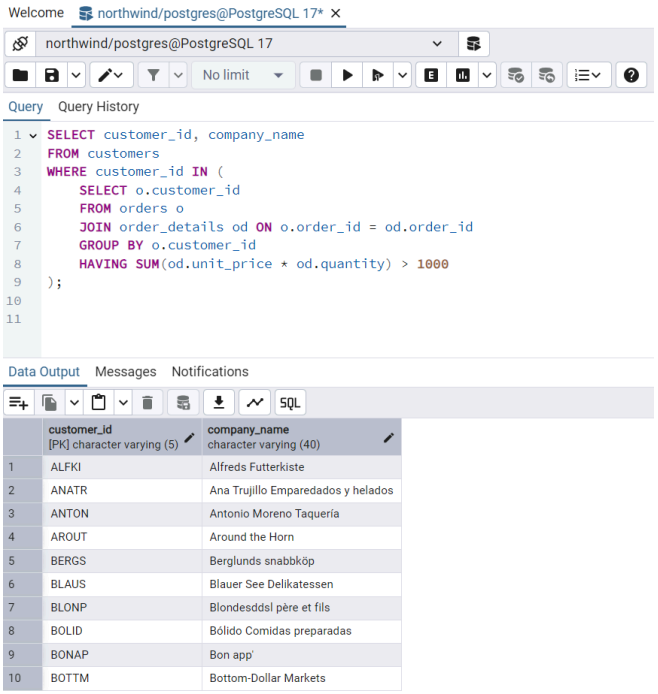
```
SELECT e.employee_id, e.first_name || ' ' || e.last_name AS
employee_name, COUNT(o.order_id) AS total_orders
FROM orders o
JOIN employees e ON o.employee_id = e.employee_id
GROUP BY e.employee_id, employee_name
ORDER BY total_orders DESC;
```



3.2. Subconsultas.

- **Clientes que han realizado pedidos superiores a $1000:**

```
SELECT customer_id, company_name
FROM customers
WHERE customer_id IN (
    SELECT o.customer_id
    FROM orders o
    JOIN order_details od ON o.order_id = od.order_id
    GROUP BY o.customer_id
    HAVING SUM(od.unit_price * od.quantity) > 1000
);
```



4. Vistas.

4.1. Creación de Vistas.

Vista de productos más vendidos:

```
SELECT p.product_id,
    p.product_name,
    c.category_name,
    sum(od.quantity) AS unidades_vendidas,
    p.units_in_stock,
    p.reorder_level,
        CASE
            WHEN p.units_in_stock <= p.reorder_level THEN 'CRÍTICO'::text
            WHEN p.units_in_stock::numeric <= (p.reorder_level::numeric * 1.5) THEN
'BAJO'::text
            ELSE 'OK'::text
        END AS estado_inventario
   FROM products p
     JOIN categories c ON p.category_id = c.category_id
     JOIN order_details od ON p.product_id = od.product_id
  GROUP BY p.product_id, p.product_name, c.category_name, p.units_in_stock,
p.reorder_level
  ORDER BY (sum(od.quantity)) DESC
 LIMIT 50;
```

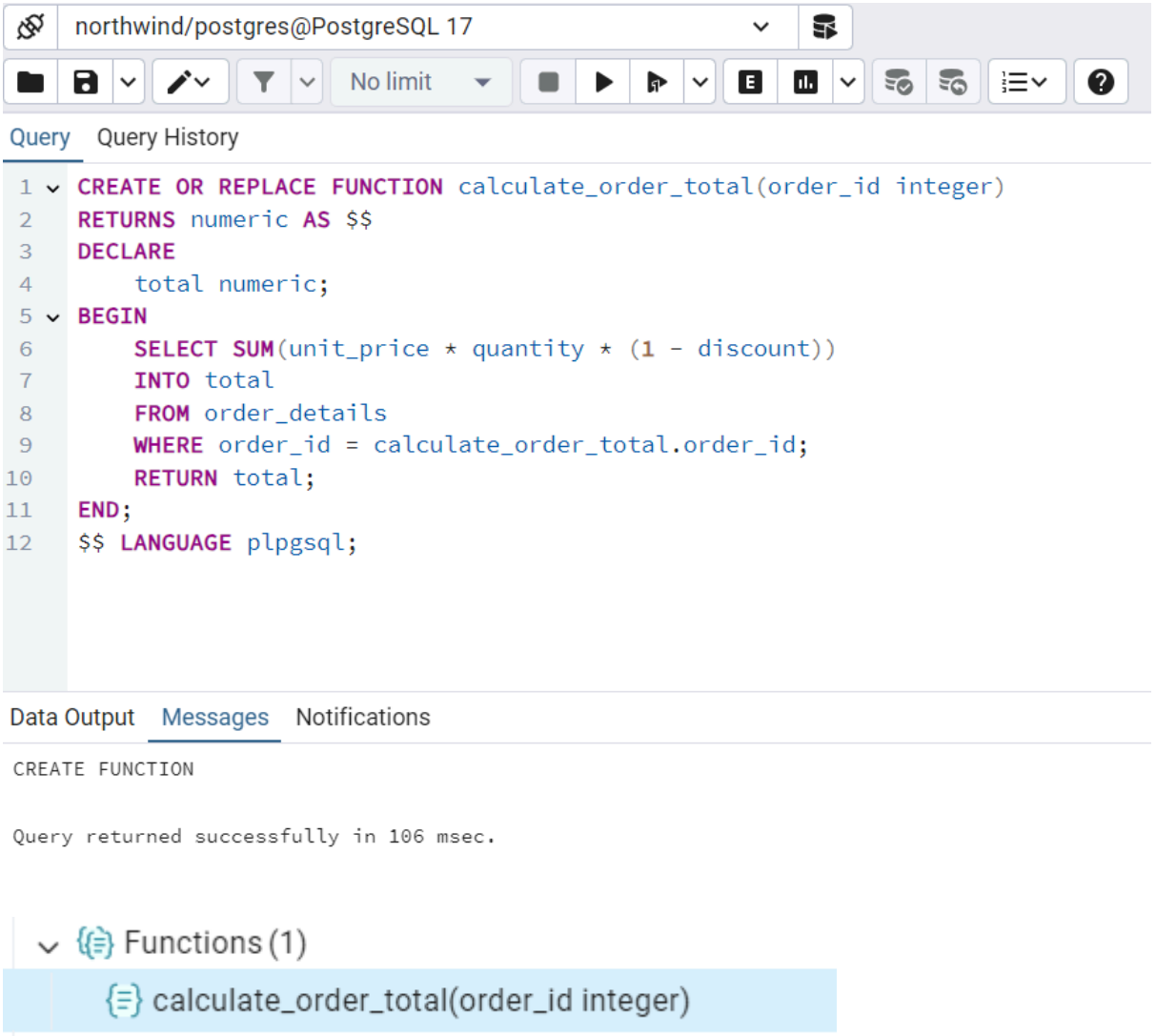
Vista de clientes con mayores compras:

```
SELECT customers.company_name AS cliente,
    sum(order_details.quantity::double precision * order_details.unit_price) AS
volumen_ventas
   FROM customers
     JOIN orders ON customers.customer_id::text = orders.customer_id::text
     JOIN order_details ON orders.order_id = order_details.order_id
  GROUP BY customers.company_name
  ORDER BY (sum(order_details.quantity::double precision * order_details.unit_price))
DESC
 LIMIT 15;
```

5. Funciones Almacenadas.

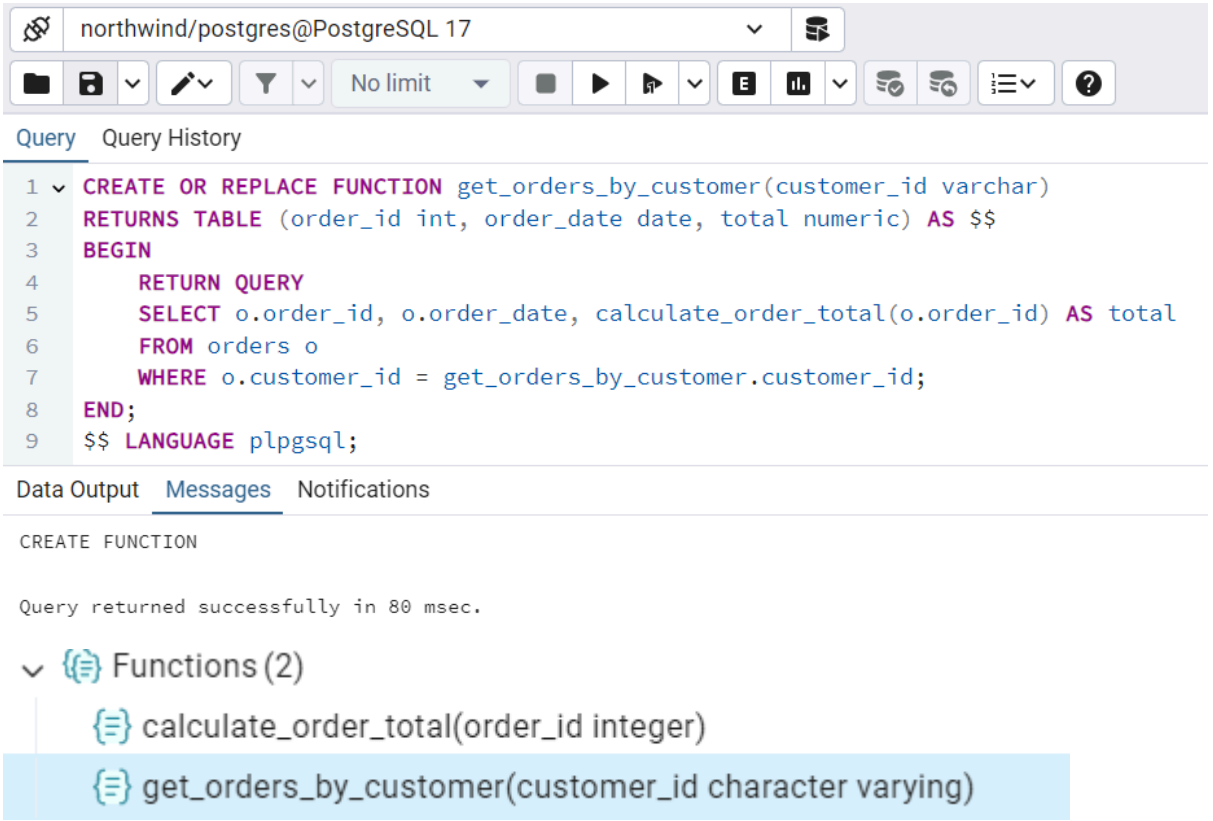
5.1. Función para Calcular Total de Pedido.

```
CREATE OR REPLACE FUNCTION calculate_order_total(order_id integer)
RETURNS numeric AS $$
DECLARE
   total numeric;
BEGIN
   SELECT SUM(unit_price * quantity * (1 - discount))
   INTO total
   FROM order_details
   WHERE order_id = calculate_order_total.order_id;
   RETURN total;
END;
$$ LANGUAGE plpgsql;
```



5.2. Función para Obtener Pedidos por Cliente.

```
CREATE OR REPLACE FUNCTION get_orders_by_customer(customer_id varchar)
RETURNS TABLE (order_id int, order_date date, total numeric) AS $$
BEGIN
   RETURN QUERY
   SELECT o.order_id, o.order_date, calculate_order_total(o.order_id) AS total
   FROM orders o
   WHERE o.customer_id = get_orders_by_customer.customer_id;
END;
$$ LANGUAGE plpgsql;
```



6. Triggers

6.1. Trigger para Actualizar Stock

```
CREATE OR REPLACE FUNCTION update_product_stock()
RETURNS TRIGGER AS $$
BEGIN
   UPDATE products
   SET units_in_stock = units_in_stock - NEW.quantity
   WHERE product_id = NEW.product_id;
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER after_order_detail_insert
AFTER INSERT ON order_details
FOR EACH ROW
EXECUTE FUNCTION update_product_stock();
```

7. Pruebas y Validación.

Probar vistas:
SELECT * FROM top_selling_products;
SELECT * FROM top_customers;

Probar funciones:
SELECT calculate_order_total(10248);
SELECT * FROM get_orders_by_customer('ALFKI');

Probar trigger (insertar un nuevo pedido y verificar stock):
INSERT INTO order_details (order_id, product_id, unit_price, quantity, discount)
VALUES (10248, 11, 14.00, 5, 0);

Verificar stock actualizado:
SELECT product_id, product_name, units_in_stock FROM products WHERE product_id = 11;

8. Agregar campo JSON en una tabla, para cumplir con el nuevo requerimiento agregamos una nueva tabla product_metadata que almacene información adicional variable de los productos en formato JSON:

Crear tabla con campo JSONB:

```
CREATE TABLE product_metadata (
   metadata_id SERIAL PRIMARY KEY,
   product_id INT NOT NULL REFERENCES products(product_id),
   attributes JSONB NOT NULL,
   last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Insertar datos de ejemplo:

```
INSERT INTO product_metadata (product_id, attributes)
VALUES
(1, '{
   "color_options": ["red", "blue", "green"],
   "weight": "0.5kg",
   "dimensions": {"length": 20, "width": 10, "height": 5},
   "compatibility": ["model-A", "model-B"],
   "warranty": {
      "period": 24,
      "type": "standard",
      "notes": "Includes parts and labor"
   }
}'),
(2, '{
   "material": "stainless steel",
   "energy_rating": "A++",
   "installation_requirements": ["220V", "dedicated circuit"],
   "accessories_included": true,
   "user_manual": "https://example.com/manuals/product2.pdf"
}');
```

Crear índices para búsquedas eficientes:

```
CREATE INDEX idx_attributes_gin ON product_metadata USING GIN (attributes);
CREATE INDEX idx_warranty_period ON product_metadata
((attributes->'warranty'->>'period'));
```

Consultas JSON de Ejemplo:

1. Productos con garantía de más de 12 meses.

```
SELECT p.product_name, pm.attributes->'warranty'->>'period' AS
warranty_months
```

```
FROM products p
JOIN product_metadata pm ON p.product_id = pm.product_id
WHERE (pm.attributes->'warranty'->>'period')::int > 12;
```

2. Buscar productos por material.

```
SELECT p.product_id, p.product_name
FROM products p
JOIN product_metadata pm ON p.product_id = pm.product_id
WHERE pm.attributes @> '{"material": "stainless steel"}';
```

3. Actualizar atributos JSON.

```
UPDATE product_metadata
SET attributes = jsonb_set(
   attributes,
   '{warranty,period}',
   '36'
)
WHERE product_id = 1;
```

4. Productos con manual de usuario disponible.

```
SELECT p.product_name, pm.attributes->>'user_manual' AS manual_url
FROM products p
JOIN product_metadata pm ON p.product_id = pm.product_id
WHERE pm.attributes ? 'user_manual';
```

5. Agregar nuevo atributo a todos los productos.

```
UPDATE product_metadata
SET attributes = attributes || '{"last_reviewed": "2023-07-15"}';
```

9. Propuesta: Sistema de Auditoría para Cambios en Pedidos.

Se procede a implementar un sistema de auditoría que registre automáticamente los cambios en los pedidos, almacenando tanto los datos anteriores como los nuevos en formato JSON. Esta solución personalizada ofrece:

1. Trazabilidad completa de modificaciones.
2. Historial de cambios para cada pedido.
3. Almacenamiento eficiente en JSON.
4. Registro de usuario y timestamp.

Tabla de Auditoría:

```
CREATE TABLE order_audit (
    audit_id SERIAL PRIMARY KEY,
    order_id INT NOT NULL REFERENCES orders(order_id),
    changed_by VARCHAR(50) NOT NULL DEFAULT CURRENT_USER,
    change_time TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    operation_type VARCHAR(10) NOT NULL CHECK (operation_type IN ('INSERT',
'UPDATE', 'DELETE')),
    old_data JSONB,
    new_data JSONB
);
```

Función para el Trigger:

```
CREATE OR REPLACE FUNCTION log_order_changes()
RETURNS TRIGGER AS $$
BEGIN
    IF (TG_OP = 'DELETE') THEN
        INSERT INTO order_audit (order_id, operation_type, old_data)
        VALUES (OLD.order_id, 'DELETE', row_to_json(OLD)::JSONB);
        RETURN OLD;
    ELSIF (TG_OP = 'UPDATE') THEN
        INSERT INTO order_audit (order_id, operation_type, old_data, new_data)
        VALUES (
            NEW.order_id,
            'UPDATE',
            jsonb_strip_nulls(row_to_json(OLD)::JSONB),
            jsonb_strip_nulls(row_to_json(NEW)::JSONB)
        );
        RETURN NEW;
    ELSIF (TG_OP = 'INSERT') THEN
        INSERT INTO order_audit (order_id, operation_type, new_data)
        VALUES (NEW.order_id, 'INSERT', row_to_json(NEW)::JSONB);
        RETURN NEW;
    END IF;
    RETURN NULL;
```

```
END;
$$ LANGUAGE plpgsql;
```

Trigger para Registro de Cambios:

```
CREATE TRIGGER orders_audit_trigger
AFTER INSERT OR UPDATE OR DELETE ON orders
FOR EACH ROW EXECUTE FUNCTION log_order_changes();
```

Casos de Uso y Consultas:

1. Verificar funcionamiento del trigger:

Actualizar un pedido:

```
UPDATE orders
SET ship_city = 'Nueva Ciudad'
WHERE order_id = 10248;
```

Insertar nuevo pedido:

```
INSERT INTO orders (order_id, customer_id, employee_id, order_date)
VALUES (11078, 'VINET', 5, '2023-07-15');
```

Eliminar un pedido (ejemplo, no ejecutar en producción):

```
DELETE FROM orders WHERE order_id = 11078;
```

2. Consultar el historial de auditoría:

Ver todos los cambios recientes:

```
SELECT
   audit_id,
   order_id,
   changed_by,
   change_time AT TIME ZONE 'UTC' AS change_time,
   operation_type,
   old_data,
   new_data
FROM order_audit
ORDER BY change_time DESC;
```

Cambios específicos en una columna:

```
SELECT
   change_time,
```

```
    changed_by,
    old_data->>'ship_city' AS old_city,
    new_data->>'ship_city' AS new_city
FROM order_audit
WHERE operation_type = 'UPDATE'
  AND (old_data->>'ship_city') IS DISTINCT FROM (new_data->>'ship_city');
```

3. Recuperar estado anterior de un pedido:

Recuperar datos antiguos de un pedido:

```
SELECT
    (old_data->>'order_id')::INT AS order_id,
    old_data->>'customer_id' AS customer_id,
    old_data->>'order_date' AS order_date,
    old_data->>'ship_city' AS ship_city
FROM order_audit
WHERE order_id = 10248
  AND operation_type = 'UPDATE'
ORDER BY change_time DESC
LIMIT 1;
```

Beneficios de esta Implementación Personalizada

1. Auditoría completa: Registra todos los cambios (inserciones, actualizaciones y eliminaciones)
2. Almacenamiento eficiente: Usa JSONB para guardar versiones completas de registros
3. Contexto de cambios: Registra quién hizo el cambio y cuándo
4. Recuperación de datos: Permite ver estados anteriores de los pedidos
5. Bajo mantenimiento: Automático, sin intervención manual
6. Flexibilidad: Puede extenderse fácilmente a otras tablas

Esta solución es especialmente útil para:

- Cumplimiento de normativas (GDPR, SOX)
- Análisis forense de datos
- Solución de problemas operativos
- Mantenimiento de historial completo de transacciones.