

LUIS DIAZ

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	Dueño
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 --
```

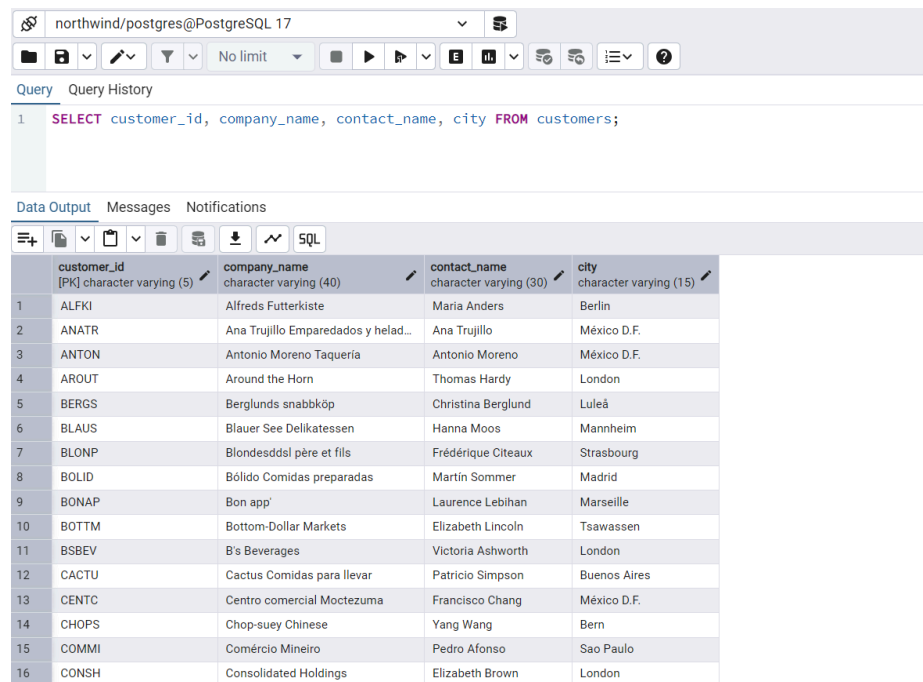
2. Consultas SQL Básicas.

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;
```

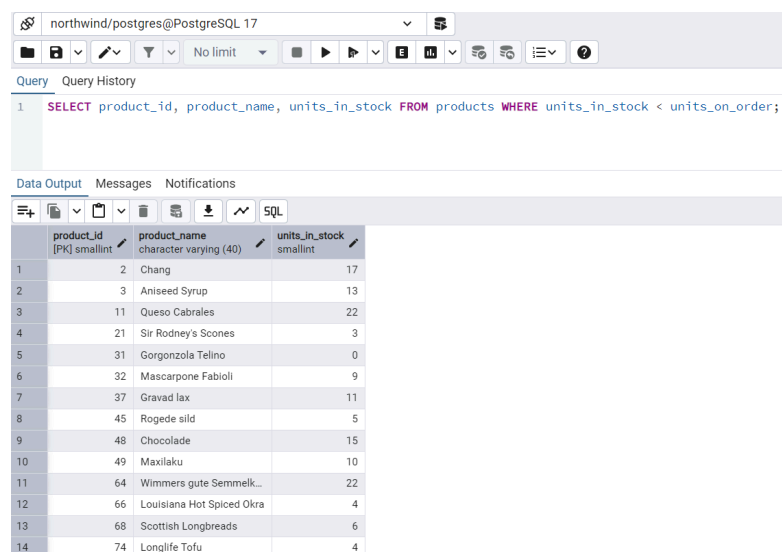


The screenshot shows a PostgreSQL query editor interface. The query entered is `SELECT customer_id, company_name, contact_name, city FROM customers;`. The results are displayed in a table with 4 columns: `customer_id` (PK), `company_name`, `contact_name`, and `city`. The table contains 16 rows of data.

	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	Blondesdssl père et fils	Frédérique Citeaux	Strasbourg
8	BOLID	Bólido Comidas preparadas	Martin 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;
```



The screenshot shows a PostgreSQL query editor interface. The query entered is `SELECT product_id, product_name, units_in_stock FROM products WHERE units_in_stock < units_on_order;`. The results are displayed in a table with 3 columns: `product_id` (PK), `product_name`, and `units_in_stock`. The table contains 14 rows of data.

	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';
```

The screenshot shows a PostgreSQL query editor with the following query:

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

The results are displayed in a table with the following columns: order_id (PK), customer_id, and order_date. The table contains 16 rows of data.

order_id	customer_id	order_date
10400	EASTC	1997-01-01
10401	RATTC	1997-01-01
10402	ERNSH	1997-01-02
10403	ERNSH	1997-01-03
10404	MAGAA	1997-01-03
10405	LINOD	1997-01-06
10406	QUEEN	1997-01-07
10407	OTTIK	1997-01-07
10408	FOLIG	1997-01-08
10409	OCEAN	1997-01-09
10410	BOTTM	1997-01-10
10411	BOTTM	1997-01-10
10412	WARTH	1997-01-13
10413	LAMAI	1997-01-14
10414	FAMIA	1997-01-14
10415	HUNGC	1997-01-15

- Productos ordenados por precio descendente

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

The screenshot shows a PostgreSQL query editor with the following query:

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

The results are displayed in a table with the following columns: product_name and unit_price. The table contains 16 rows of data, ordered by unit price in descending order.

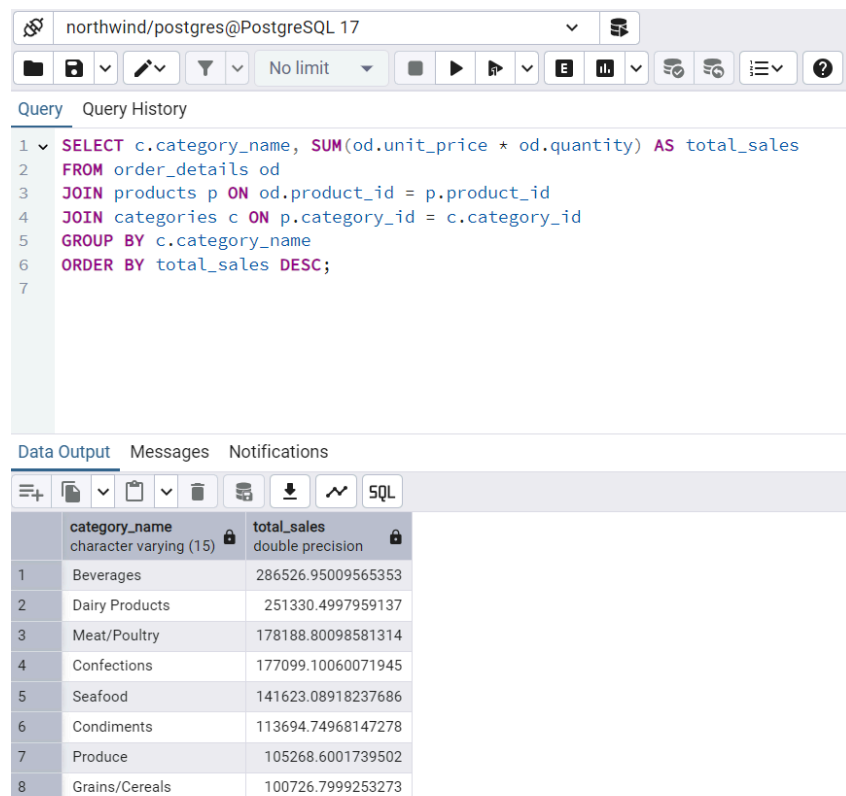
product_name	unit_price
Côte de Blaye	263.5
Thüringer Rostbratwurst	123.79
Mishi Kobe Niku	97
Sir Rodney's Marmalade	81
Carnarvon Tigers	62.5
Raclette Courdavault	55
Manjimup Dried Apples	53
Tarte au sucre	49.3
Ipoh Coffee	46
Rössle Sauerkraut	45.6
Schoggi Schokolade	43.9
Veggie-spread	43.9
Northwoods Cranberry Sauce	40
Alice Mutton	39
Gnocchi di nonna Alice	38
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;
```



The screenshot shows a PostgreSQL query editor interface. At the top, the connection is 'northwind/postgres@PostgreSQL 17'. Below the toolbar, the query is entered in the 'Query' tab. The results are displayed in the 'Data Output' tab as a table with 8 rows and 2 columns: 'category_name' and 'total_sales'.

	category_name	total_sales
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;
```

Query Query History

```

1 SELECT e.employee_id, e.first_name || ' ' || e.last_name AS employee_name, COUNT(o.order_id) AS total_orders
2 FROM orders o
3 JOIN employees e ON o.employee_id = e.employee_id
4 GROUP BY e.employee_id, employee_name
5 ORDER BY total_orders DESC;
6

```

Data Output Messages Notifications

	employee_id [PK] smallint	employee_name text	total_orders bigint
1	4	Margaret Peacock	156
2	3	Janet Leverling	127
3	1	Nancy Davolio	123
4	8	Laura Callahan	104
5	2	Andrew Fuller	96
6	7	Robert King	72
7	6	Michael Suyama	67
8	9	Anne Dodsworth	43
9	5	Steven Buchanan	42

3.2. Subconsultas.

- **Cientes 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
);

```

Welcome northwind/postgres@PostgreSQL 17 x

northwind/postgres@PostgreSQL 17

Query Query History

```

1 SELECT customer_id, company_name
2 FROM customers
3 WHERE customer_id IN (
4     SELECT o.customer_id
5     FROM orders o
6     JOIN order_details od ON o.order_id = od.order_id
7     GROUP BY o.customer_id
8     HAVING SUM(od.unit_price * od.quantity) > 1000
9 );
10
11

```

Data Output Messages Notifications

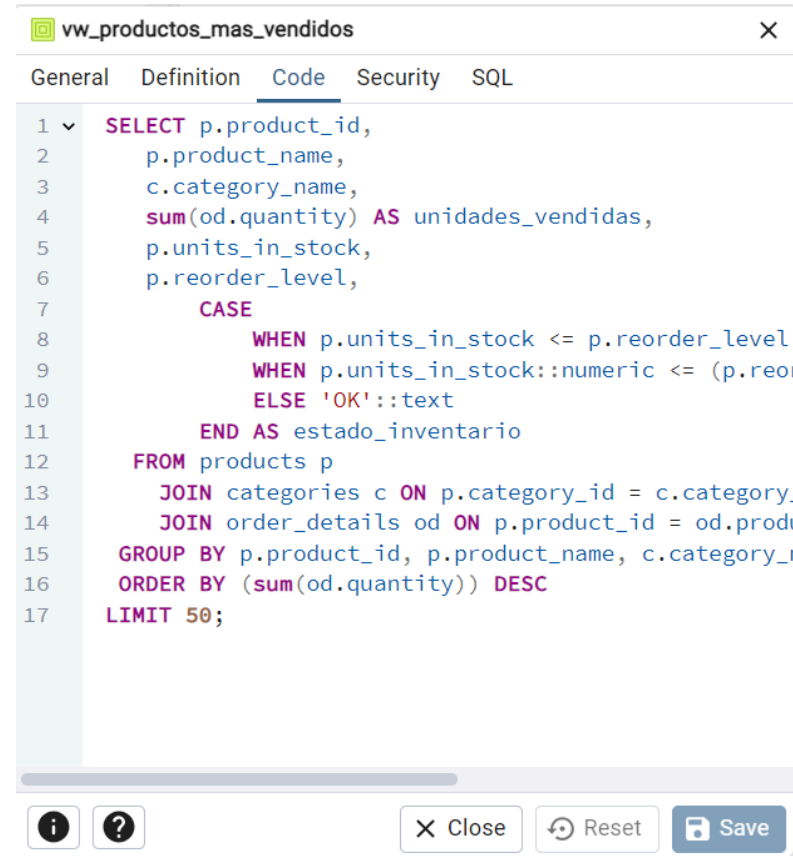
	customer_id [PK] character varying (5)	company_name character varying (40)
1	ALFKI	Alfreds Futterkiste
2	ANATR	Ana Trujillo Emparedados y helados
3	ANTON	Antonio Moreno Taquería
4	AROUT	Around the Horn
5	BERGS	Berglunds snabbköp
6	BLAUS	Blauer See Delikatessen
7	BLONP	Blondesddsl père et fils
8	BOLID	Bólido Comidas preparadas
9	BONAP	Bon app'
10	BOTTM	Bottom-Dollar Markets

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;
```

 vw_top_clientes

General Definition Code Security SQL

```
1  SELECT customers.company_name AS cliente,  
2      sum(order_details.quantity::double precision * or  
3  FROM customers  
4      JOIN orders ON customers.customer_id::text = ord  
5      JOIN order_details ON orders.order_id = order_de  
6  GROUP BY customers.company_name  
7  ORDER BY (sum(order_details.quantity::double precis  
8  LIMIT 15;
```



✕ Close

↺ Reset

💾 Save

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;
```

The screenshot shows a PostgreSQL client interface with the following components:

- Toolbar:** Includes icons for file operations, query execution, and settings. A dropdown menu shows "No limit".
- Query Editor:** Displays the SQL code for creating the function `calculate_order_total`. The code is as follows:

```
1 CREATE OR REPLACE FUNCTION calculate_order_total(order_id integer)
2 RETURNS numeric AS $$
3 DECLARE
4     total numeric;
5 BEGIN
6     SELECT SUM(unit_price * quantity * (1 - discount))
7     INTO total
8     FROM order_details
9     WHERE order_id = calculate_order_total.order_id;
10    RETURN total;
11 END;
12 $$ LANGUAGE plpgsql;
```
- Messages Tab:** Shows the execution result: "CREATE FUNCTION". Below this, it states "Query returned successfully in 106 msec."
- Database Explorer:** Shows a tree view with "Functions (1)" expanded, listing the function `calculate_order_total(order_id integer)`.

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;
```

The screenshot shows a PostgreSQL client interface with the following components:

- Top Bar:** Displays the connection name 'northwind/postgres@PostgreSQL 17' and a toolbar with icons for file operations, query execution, and settings.
- Query Editor:** Contains the SQL code for creating the function 'get_orders_by_customer'. The code is syntax-highlighted and numbered from 1 to 9.
- Messages Tab:** Shows the execution results, including the message 'CREATE FUNCTION' and 'Query returned successfully in 80 msec.'
- Functions List:** A section titled 'Functions (2)' showing the two functions created: 'calculate_order_total(order_id integer)' and 'get_orders_by_customer(customer_id character varying)'. The latter is highlighted with a blue background.

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.