
















## Índice

- Proyecto Northwind PostgreSQL - Modificado
  -  Descripción del Proyecto
    -  Nuevas Funcionalidades
  -  Tecnologías
  -  Estructura del Repositorio
  -  Instalación Rápida
    - Prerrequisitos
    - Instalación en 3 pasos
    - Alternativa con pgAdmin
  -  Funcionalidades Principales
    - 1. Modificación de la tabla Products
    - 2. Propuesta del alumno
  -  Nuevas Tablas Añadidas
  -  Vistas Creadas
  -  Funciones y Triggers
  -  Datos de Prueba
  -  Validar Instalación
  -  Especificaciones Técnicas
  -  Información Académica
  -  Soporte
  -  Objetivos de Aprendizaje Demostrados
    - Sistema de Categorías Jerárquica
      - ¿Cómo funciona?
      - Scripts de implementación
      - Ejemplo de consulta
    - Control de Stock Avanzado
      - Scripts de implementación
      - Consulta de productos con stock bajo
    - Descuentos por Volumen
      - ¿Cómo funciona?
      - Scripts de implementación
      - Ejemplo de consulta
    - Auditoría Completa de Cambios en Productos
      - ¿Cómo funciona?
      - Scripts de implementación
      - Ejemplo de consulta
    - Vistas de Análisis
      - Scripts de implementación
      - Ejemplo de consulta
    - Triggers Inteligentes
      - ¿Cómo funcionan?
      - Scripts de implementación

# Proyecto Northwind PostgreSQL - Modificado

---

Este repositorio contiene una versión modificada de la base de datos Northwind para PostgreSQL, desarrollada como proyecto de curso con nuevas funcionalidades y mejoras.

## Descripción del Proyecto

La base de datos Northwind ha sido extendida con las siguientes mejoras:

### Nuevas Funcionalidades

- **Sistema de Categorías Jerárquicas:** Subcategorías para mejor organización
- **Control de Stock Avanzado:** Alertas automáticas y stock mínimo
- **Descuentos por Volumen:** Sistema automatizado de descuentos
- **Auditoría Completa:** Registro de cambios en productos
- **Vistas de Análisis:** Reportes de ventas y productos
- **Triggers Inteligentes:** Automatización de procesos
- **Historial de Precios:** Registro automático de cambios de precio
- **Validación de Emails:** Restricción para asegurar emails válidos en clientes
- **Soft Delete:** Borrado lógico de productos
- **Actualización automática de stock tras pedido**
- **Índices optimizados y uso de JSONB**

## Tecnologías

- **PostgreSQL 17.5**
- **pgAdmin** (opcional)
- **SQL Dump** para instalación rápida

## Estructura del Repositorio

```
northwind-postgres-modificado/
├── README.md                # Este archivo
├── northwind_modificado.sql  # ★ DUMP COMPLETO DE LA BD
├── docs/
│   ├── INSTALACION.md      # Guía de instalación
│   ├── FUNCIONALIDADES.md  # Documentación de mejoras
│   └── CONSULTAS_EJEMPLO.md # Ejemplos de uso
└── screenshots/
    ├── diagrama_er.png     # Diagrama actualizado
    └── consultas_ejemplo.png # Capturas de pantalla
```

## Instalación Rápida

### Prerrequisitos

- PostgreSQL 17

- Cliente psql o pgAdmin

## Instalación en 3 pasos

### 1. Clonar repositorio

```
git clone https://github.com/JuanHobb/NORTHWIND_MODIFICADO.git
cd northwind-postgres-modificado
```

### 2. Crear base de datos

```
createdb northwind_curso
```

### 3. Restaurar dump completo

```
psql -d northwind_curso -f northwind_modificado.sql
```

¡Y listo! La base de datos estará completamente configurada con datos de ejemplo.

## Alternativa con pgAdmin

1. Crear nueva base de datos llamada `northwind_curso`
2. Click derecho → Restore
3. Seleccionar archivo `northwind_modificado.sql`
4. Ejecutar

## Funcionalidades Principales

### 1. Modificación de la tabla Products

Objetivo: Añadir un campo JSON para almacenar atributos dinámicos de productos (ej: especificaciones técnicas, metadatos).

1. Añadir la columna `caracteristicas_json` en la tabla Products.
2. Rellenar la nueva columna con datos JSON (ejemplos de "categoria" y "subcategoria").
3. Ejecutar consultas sobre los datos JSON para filtrar productos por categoría o subcategoría.

---

### 2. Propuesta del alumno

En el mundo actual, un comercio necesita que su base de datos sea mucho más que un simple almacén de información; debe ser una herramienta que facilite la gestión diaria y ayude a tomar mejores decisiones. Por eso, las mejoras que se han implementado buscan justamente hacer que la base de datos sea más flexible, segura y eficiente.

Por ejemplo, al usar un campo JSONB para los atributos de los productos, se consigue que cada artículo pueda tener características propias sin complicar la estructura de la base de datos. Esto es muy útil cuando se manejan productos con muchas variantes, ya que permite adaptarse rápidamente a las necesidades del negocio sin grandes cambios técnicos.

También se optó por un borrado lógico en lugar de eliminar productos definitivamente. Esto no solo ayuda a mantener un historial valioso, sino que también permite recuperar información si se comete algún error, algo que siempre es importante en la gestión diaria.

La organización de las categorías en forma jerárquica facilita que tanto los clientes como el equipo interno puedan encontrar productos con mayor facilidad. Esta estructura refleja mejor la realidad del catálogo y mejora la experiencia de navegación.

En cuanto al control de stock, automatizar las alertas y actualizaciones evita sorpresas desagradables como vender productos agotados. Esto no solo mejora la confianza del cliente, sino que también libera al equipo de tareas repetitivas y propensas a errores.

La implementación de descuentos por volumen automatizados ayuda a incentivar compras mayores y asegura que las promociones se apliquen de forma justa y consistente, lo que es clave para mantener la satisfacción del cliente.

Registrar todos los cambios en los productos, incluyendo los datos anteriores y nuevos, aporta una capa extra de seguridad y transparencia. Esto facilita detectar cualquier error o modificación inesperada, algo fundamental para mantener la integridad de la información.

Además, llevar un historial de precios permite entender cómo evolucionan a lo largo del tiempo y tomar decisiones más informadas sobre ofertas y estrategias comerciales.

Validar automáticamente los correos electrónicos de los clientes mejora la calidad de los datos y evita problemas en las campañas de comunicación, lo que se traduce en una mejor relación con los clientes.

Por último, actualizar el stock automáticamente tras cada pedido y contar con reportes rápidos y claros ayuda a mantener la información siempre actualizada y facilita la toma de decisiones en tiempo real.

En conjunto, estas mejoras no solo optimizan el funcionamiento técnico de la base de datos, sino que también aportan valor real al negocio, haciendo que la gestión sea más ágil, segura y orientada a las necesidades reales del comercio.

Como parte del ejercicio de evaluación y mejora, se han implementado propuestas personales que demuestran la capacidad de extender y optimizar una base de datos real para necesidades empresariales modernas. Estas mejoras avanzadas aplican buenas prácticas de diseño, automatización y análisis en PostgreSQL.

### **Resumen de propuestas implementadas:**

- **Campo JSONB en productos:**  
Permite almacenar atributos dinámicos y realizar búsquedas flexibles y optimizadas mediante índices GIN.
- **Soft delete en productos:**  
Borrado lógico para mantener el histórico de productos sin eliminarlos físicamente.

- **Sistema de categorías jerárquicas:**  
Subcategorías enlazadas a categorías principales para una organización avanzada del catálogo.
- **Control de stock avanzado:**  
Alertas automáticas y triggers para gestionar inventario y evitar roturas de stock.
- **Descuentos por volumen:**  
Tabla y función para aplicar descuentos automáticos según la cantidad comprada.
- **Auditoría completa:**  
Trigger y tabla para registrar todos los cambios realizados en productos, incluyendo los datos antiguos y nuevos en formato JSONB.
- **Historial de precios de productos:**  
Trigger y tabla para registrar cada cambio de precio de los productos.
- **Validación de emails en clientes:**  
Restricción CHECK para asegurar que los emails introducidos sean válidos.



Validación de emails




*Implementación de restricción para emails válidos en clientes*



- **Descuento automático de stock tras cada pedido:**  
Trigger que descuenta automáticamente el stock al registrar un pedido.
- **Vistas materializadas y de análisis:**  
Para obtener reportes rápidos y facilitar la toma de decisiones empresariales.
- **Índices optimizados:**  
Uso de índices compuestos y GIN para mejorar el rendimiento de las consultas.
- **Triggers inteligentes:**  
Automatización de procesos críticos como auditoría, alertas y control de inventario.

Estas propuestas reflejan la aplicación de conocimientos avanzados en administración y desarrollo de bases de datos PostgreSQL, orientados a la mejora continua y la adaptación a escenarios empresariales reales.





## Nuevas Tablas Añadidas

- **subcategories** - Categorías jerárquicas
-  Tabla de subcategorías  
*Implementación del sistema de categorías jerárquicas*
- **volume\_discounts** - Descuentos por cantidad
-  Tabla de descuentos por volumen  
*Implementación del sistema de descuentos por volumen*
- **product\_audit** - Auditoría de cambios
-  Tabla de auditoría de productos  
*Implementación del sistema de auditoría de productos*
- **stock\_alerts** - Alertas de inventario

-  Tabla de alertas de inventario  
*Implementación del sistema de alertas de inventario*
- `product_price_history` - Historial de precios
-  Tabla de historial de precios  
*Implementación del sistema de historial de precios*

## Vistas Creadas

- `productos_stock_bajo` - Control de inventario
- `ventas_mensuales` - Análisis temporal
- `top_productos_vendidos` - Ranking de productos
- `analisis_clientes` - Segmentación de clientes
-  Vistas de análisis  
*Vista de top productos vendidos y análisis de clientes*
- `ventas_mensuales_mat` - Vista materializada de ventas mensuales
-  Vista materializada  
*Vista materializada para reportes de ventas mensuales*

## Funciones y Triggers

- **Auditoría automática** en cambios de productos
- **Alertas de stock** cuando baja del mínimo
- **Cálculo de descuentos** por volumen de compra
- **Actualización automática** de timestamps
- **Descuento automático de stock tras pedido**
- **Historial de precios** mediante triggers

## Datos de Prueba

El dump incluye:

- Base Northwind completa original
- 8 subcategorías de ejemplo
- 8 reglas de descuento por volumen
- Configuración de stock mínimo
- Alertas de ejemplo generadas

## Validar Instalación

Después de restaurar, ejecuta estas consultas para verificar:

```
-- Verificar nuevas tablas
SELECT count(*) FROM subcategories;           -- Debe mostrar 8
SELECT count(*) FROM volume_discounts;       -- Debe mostrar 8
SELECT count(*) FROM stock_alerts;           -- Debe mostrar varias

-- Probar vistas
SELECT count(*) FROM productos_stock_bajo;
SELECT count(*) FROM ventas_mensuales;
```

```
-- Probar función  
SELECT calcular_descuento_volumen(1, 100); -- Debe mostrar 10.00
```

## Especificaciones Técnicas

- **Versión PostgreSQL:** 12+
- **Tamaño del dump:** ~500KB
- **Total tablas:** 17 (13 originales + 4 nuevas)
- **Total vistas:** 4
- **Total funciones:** 3
- **Triggers:** 1 principal con múltiples eventos

## Información Académica

- **Curso:** Bases de Datos Avanzadas
- **Institución:** Urko Servicios de Prevención S. Coop.
- **Autor:** Juan de la Morena Marzalo
- **Fecha:** Mayo 2025

## Soporte

Si tienes problemas con la instalación:

1. Verifica que PostgreSQL esté corriendo
2. Asegúrate de tener permisos para crear BD
3. Revisa que el archivo SQL esté completo
4. Consulta los logs de PostgreSQL para errores

## Objetivos de Aprendizaje Demostrados

- ☒ Modificación de esquemas existentes
- ☒ Creación de tablas relacionadas
- ☒ Implementación de triggers
- ☒ Desarrollo de vistas complejas
- ☒ Funciones en PL/pgSQL
- ☒ Optimización con índices
- ☒ Sistemas de auditoría
- ☒ Generación de dumps

---

**Nota:** Este proyecto demuestra conocimientos avanzados en PostgreSQL aplicados sobre la conocida base de datos Northwind, añadiendo funcionalidades empresariales reales.

### Sistema de Categorías Jerárquica

Northwind originalmente solo permite asociar productos a una categoría principal. Para mejorar la organización, hemos implementado **subcategorías** que permiten una estructura jerárquica flexible.

## ¿Cómo funciona?

- **categories:** Tabla original, contiene las categorías principales.
- **subcategories:** Nueva tabla, cada subcategoría pertenece a una categoría.
- **products:** Ahora cada producto puede asociarse a una subcategoría.

## Scripts de implementación

```
-- 1. Crear la tabla de subcategorías
CREATE TABLE subcategories (
    subcategory_id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    category_id INT NOT NULL,
    FOREIGN KEY (category_id) REFERENCES categories(category_id)
);

-- 2. Añadir columna subcategory_id a products y crear la relación
ALTER TABLE products
ADD COLUMN subcategory_id INT;

ALTER TABLE products
ADD CONSTRAINT fk_products_subcategories
FOREIGN KEY (subcategory_id) REFERENCES subcategories(subcategory_id);

-- 3. Insertar ejemplos de subcategorías
INSERT INTO subcategories (name, category_id) VALUES
('Refrescos', 1),
('Zumos', 1),
('Quesos duros', 2),
('Quesos blandos', 2),
('Café', 1),
('Tés', 1),
('Embutidos', 3),
('Carne curadas', 3);

-- 4. Asignar subcategorías a productos (ejemplo)
UPDATE products SET subcategory_id = 1 WHERE product_id = 1; -- Producto 1 a 'Refrescos'
UPDATE products SET subcategory_id = 2 WHERE product_id = 2; -- Producto 2 a 'Zumos'
```

## Ejemplo de consulta

```
SELECT p.product_id, p.product_name, s.name AS subcategoria, c.category_name
FROM products p
JOIN subcategories s ON p.subcategory_id = s.subcategory_id
JOIN categories c ON s.category_id = c.category_id
ORDER BY c.category_name, s.name, p.product_name;
```



**Ventajas:**

- Permite filtrar productos por subcategoría o categoría principal.
- Mejora la organización y el análisis de inventario.

**Control de Stock Avanzado**

Se ha implementado un sistema de alertas automáticas para productos cuyo stock cae por debajo del mínimo definido.

**Scripts de implementación**

```
ALTER TABLE products
ADD COLUMN min_stock INT DEFAULT 10;

CREATE TABLE stock_alerts (
    alert_id SERIAL PRIMARY KEY,
    product_id INT NOT NULL,
    alert_date TIMESTAMP DEFAULT now(),
    current_stock INT,
    min_stock INT,
    resolved BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

CREATE OR REPLACE FUNCTION check_stock_alert()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.units_in_stock < NEW.min_stock THEN
        INSERT INTO stock_alerts(product_id, current_stock, min_stock)
        VALUES (NEW.product_id, NEW.units_in_stock, NEW.min_stock);
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_stock_alert
AFTER UPDATE OF units_in_stock ON products
FOR EACH ROW
EXECUTE FUNCTION check_stock_alert();
```

**Consulta de productos con stock bajo**

```
SELECT p.product_id, p.product_name, p.units_in_stock, p.min_stock
FROM products p
WHERE p.units_in_stock < p.min_stock;
```

**Ventajas:**

- Automatiza la gestión de inventario.
- Permite actuar rápidamente ante posibles roturas de stock.

## Descuentos por Volumen

Se ha implementado un sistema de descuentos automáticos según la cantidad comprada de cada producto.

### ¿Cómo funciona?

- **volume\_discounts**: Nueva tabla donde se definen los descuentos por cantidad mínima para cada producto.
- **Función calcular\_descuento\_volumen**: Devuelve el porcentaje de descuento aplicable según el producto y la cantidad.

### Scripts de implementación

```
-- 1. Crear la tabla de descuentos por volumen
CREATE TABLE volume_discounts (
    discount_id SERIAL PRIMARY KEY,
    product_id INT NOT NULL,
    min_quantity INT NOT NULL,
    discount_percent NUMERIC(5,2) NOT NULL,
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);

-- 2. Insertar reglas de ejemplo
INSERT INTO volume_discounts (product_id, min_quantity, discount_percent) VALUES
(1, 10, 5.00),    -- 5% de descuento si compras 10 o más del producto 1
(1, 50, 10.00),   -- 10% si compras 50 o más del producto 1
(2, 20, 7.50);    -- 7.5% si compras 20 o más del producto 2

-- 3. Crear función para calcular el descuento
CREATE OR REPLACE FUNCTION calcular_descuento_volumen(pid INT, cantidad INT)
RETURNS NUMERIC AS $$
DECLARE
    descuento NUMERIC := 0;
BEGIN
    SELECT COALESCE(MAX(discount_percent), 0)
    INTO descuento
    FROM volume_discounts
    WHERE product_id = pid AND cantidad >= min_quantity;
    RETURN descuento;
END;
$$ LANGUAGE plpgsql;
```

### Ejemplo de consulta

```
SELECT od.order_id, od.product_id, od.quantity,
       calcular_descuento_volumen(od.product_id, od.quantity) AS
```

```
descuento_aplicado
FROM order_details od
WHERE od.quantity >= 10;
```

### Ventajas:

- Automatiza la aplicación de descuentos por cantidad.
- Facilita la gestión de promociones y ventas al por mayor.

## Auditoría Completa de Cambios en Productos

Se ha implementado un sistema de auditoría que registra automáticamente los cambios realizados en la tabla de productos.

### ¿Cómo funciona?

- **product\_audit:** Nueva tabla donde se almacenan los cambios (antes y después) de cada producto.
- **Trigger y función:** Cada vez que se actualiza un producto, se guarda un registro en la auditoría.

### Scripts de implementación

```
-- 1. Crear la tabla de auditoría
CREATE TABLE product_audit (
    audit_id SERIAL PRIMARY KEY,
    product_id INT,
    old_data JSONB,
    new_data JSONB,
    changed_at TIMESTAMP DEFAULT now()
);

-- 2. Crear función de auditoría
CREATE OR REPLACE FUNCTION audit_product_changes()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO product_audit(product_id, old_data, new_data)
    VALUES (
        NEW.product_id,
        to_jsonb(OLD),
        to_jsonb(NEW)
    );
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- 3. Crear trigger para la auditoría
CREATE TRIGGER trg_audit_products
AFTER UPDATE ON products
FOR EACH ROW
EXECUTE FUNCTION audit_product_changes();
```

## Ejemplo de consulta

```
SELECT audit_id, product_id, changed_at, old_data, new_data
FROM product_audit
ORDER BY changed_at DESC;
```

## Ventajas:

- Permite rastrear todos los cambios realizados en los productos.
- Facilita la recuperación de información histórica y la trazabilidad.

## Vistas de Análisis

Se han creado varias vistas para facilitar el análisis de ventas, inventario y clientes.

## Scripts de implementación

```
-- 1. Vista de productos con stock bajo
CREATE OR REPLACE VIEW productos_stock_bajo AS
SELECT product_id, product_name, units_in_stock, min_stock
FROM products
WHERE units_in_stock < min_stock;

-- 2. Vista de ventas mensuales por año
CREATE OR REPLACE VIEW ventas_mensuales AS
SELECT
    DATE_TRUNC('month', o.order_date) AS mes,
    SUM(od.quantity * od.unit_price * (1 - od.discount)) AS total_ventas
FROM orders o
JOIN order_details od ON o.order_id = od.order_id
GROUP BY mes
ORDER BY mes;

-- 3. Vista de top productos vendidos
CREATE OR REPLACE VIEW top_productos_vendidos AS
SELECT
    p.product_id,
    p.product_name,
    SUM(od.quantity) AS total_vendido
FROM products p
JOIN order_details od ON p.product_id = od.product_id
GROUP BY p.product_id, p.product_name
ORDER BY total_vendido DESC
LIMIT 10;

-- 4. Vista de análisis de clientes por ventas
CREATE OR REPLACE VIEW analisis_clientes AS
SELECT
    c.customer_id,
    c.company_name,
```

```
COUNT(o.order_id) AS total_pedidos,  
SUM(od.quantity * od.unit_price * (1 - od.discount)) AS total_gastado  
FROM customers c  
JOIN orders o ON c.customer_id = o.customer_id  
JOIN order_details od ON o.order_id = od.order_id  
GROUP BY c.customer_id, c.company_name  
ORDER BY total_gastado DESC;
```

### Ejemplo de consulta

```
SELECT * FROM top_productos_vendidos;
```

### Ventajas:

- Facilitan la toma de decisiones empresariales.
- Permiten obtener reportes rápidos sin escribir consultas complejas.
- Mejoran la visibilidad sobre ventas, inventario y clientes.

### Triggers Inteligentes

Se han implementado triggers que automatizan tareas críticas en la base de datos, como la auditoría de cambios y la gestión de alertas de stock.

### ¿Cómo funcionan?

- **Trigger de auditoría:** Cada vez que se actualiza un producto, se registra automáticamente el cambio en la tabla `product_audit`.
- **Trigger de alertas de stock:** Cuando el stock de un producto baja del mínimo, se inserta automáticamente una alerta en la tabla `stock_alerts`.

### Scripts de implementación

```
-- Trigger de auditoría (ya documentado arriba)  
CREATE TRIGGER trg_audit_products  
AFTER UPDATE ON products  
FOR EACH ROW  
EXECUTE FUNCTION audit_product_changes();  
  
-- Trigger de alertas de stock (ya documentado arriba)  
CREATE TRIGGER trg_stock_alert  
AFTER UPDATE OF units_in_stock ON products  
FOR EACH ROW  
EXECUTE FUNCTION check_stock_alert();
```

### Ventajas:

- Automatizan la gestión de eventos importantes sin intervención manual.
- Mejoran la integridad y trazabilidad de los datos.
- Permiten reaccionar en tiempo real ante cambios críticos en la base de datos.

**Nota sobre vistas materializadas:**

La vista `ventas_mensuales_mat` es una **vista materializada**.

En pgAdmin, no aparece en la carpeta "Views" sino en la carpeta "**Materialized Views**" dentro del esquema (normalmente `public`).

Las vistas materializadas almacenan los datos físicamente y deben refrescarse manualmente para actualizar su contenido.

- Para ver sus datos: haz clic derecho sobre la vista en "Materialized Views" y selecciona "View/Edit Data".
- Para actualizarla (refrescar los datos): ejecuta

```
REFRESH MATERIALIZED VIEW ventas_mensuales_mat;
```

- Si necesitas eliminarla para volver a crearla:

```
DROP MATERIALIZED VIEW ventas_mensuales_mat;
```

Si intentas crearla de nuevo sin borrarla antes, PostgreSQL mostrará un error de "ya existe".