

LABORATORIO INTEGRADOR: IMPLEMENTACIÓN DE CONTROL DE TRANSACCIONES

ESCENARIO

Una empresa de comercio electrónico necesita implementar un sistema de gestión de pedidos y stock que maneje operaciones concurrentes de múltiples usuarios. El sistema debe garantizar la consistencia de datos cuando varios clientes intenten comprar productos simultáneamente, evitando ventas de productos sin stock y asegurando que los pagos se procesen correctamente.

DESCRIPCIÓN DEL PROBLEMA

Usted debe implementar una base de datos que gestione las siguientes operaciones críticas del negocio:

1. **Registro de productos** con control de inventario
2. **Procesamiento de pedidos** con validación de stock disponible
3. **Gestión de pagos** vinculados a pedidos
4. **Actualización automática de inventario** al confirmar ventas
5. **Manejo de cancelaciones** con devolución de stock

El sistema debe manejar correctamente situaciones donde:

- Múltiples usuarios intentan comprar el último producto disponible
- Un pago falla después de reservar productos
- Se intenta procesar un pedido sin stock suficiente
- Ocurren errores durante el procesamiento que requieren reversión completa

ESTRUCTURA DE DATOS REQUERIDA

Debe crear las siguientes tablas con sus respectivas relaciones y

restricciones: **Tabla: productos**

- Código de producto (clave primaria)
- Nombre del producto
- Descripción
- Precio unitario
- Stock disponible
- Stock mínimo de alerta

- Estado (activo/inactivo)
- Fecha de última actualización

Tabla: clientes

- ID de cliente (clave primaria)
- Nombre completo
- Correo electrónico (único)
- Teléfono
- Dirección de envío
- Fecha de registro

Tabla: pedidos

- ID de pedido (clave primaria)
- ID de cliente (clave foránea)
- Fecha del pedido
- Estado del pedido (pendiente, confirmado, enviado, cancelado)
- Monto total
- Fecha de última actualización

Tabla: detalle_pedido

- ID de detalle (clave primaria)
- ID de pedido (clave foránea)
- Código de producto (clave foránea)
- Cantidad solicitada
- Precio unitario al momento de la compra
- Subtotal

Tabla: pagos

- ID de pago (clave primaria)
- ID de pedido (clave foránea)
- Método de pago
- Monto pagado
- Fecha y hora del pago
- Estado del pago (procesando, aprobado, rechazado)

- Referencia de transacción

Tabla: historial_stock

- ID de registro (clave primaria)
- Código de producto (clave foránea)
- Tipo de movimiento (entrada, salida, ajuste)
- Cantidad
- Stock anterior
- Stock nuevo
- ID de pedido relacionado (si aplica)
- Fecha y hora del movimiento
- Usuario que realizó el movimiento

REQUERIMIENTOS FUNCIONALES

PARTE 1: CREACIÓN Y CONFIGURACIÓN DE LA BASE DE DATOS (4

puntos) Tarea 1.1: Crear la base de datos llamada ecommerce_lab

Tarea 1.2: Implementar todas las tablas especificadas con:

- Claves primarias y foráneas correctamente definidas
- Restricciones de integridad (NOT NULL, UNIQUE, CHECK donde corresponda) • Valores por defecto apropiados
- Índices en columnas que se consultarán frecuentemente

Tarea 1.3: Insertar datos de prueba:

- Mínimo 10 productos con diferentes niveles de stock
- Al menos 5 clientes
- Incluir productos con stock bajo (1-2 unidades) para probar concurrencia

PISTAS:

- Considere qué campos deberían tener restricciones CHECK (ejemplo: stock no puede ser negativo, precios deben ser positivos)
- Piense en qué campos se usarán en búsquedas y filtros para crear índices
- Los stocks iniciales deben ser variados para permitir diferentes escenarios de prueba

PARTE : IMPLEMENTACIÓN DE PROCEDIMIENTOS DE NEGOCIO (8

puntos) Tarea 2.1: Crear una función crear_pedido que:

- Reciba como parámetros: ID del cliente, lista de productos (código y cantidad)
- Verifique que todos los productos existan y estén activos
- Valide que haya stock suficiente para cada producto solicitado
- Reserve el stock (reduzca el inventario) de todos los productos
- Cree el registro del pedido con estado 'pendiente'
- Inserte los detalles del pedido
- Registre los movimientos en el historial de stock
- Calcule y guarde el monto total del pedido
- Retorne el ID del pedido creado o NULL si falla

Tarea 2.2: Crear una función procesar_pago que:

- Reciba como parámetros: ID del pedido, método de pago, referencia de transacción
- Valide que el pedido exista y esté en estado 'pendiente'
- Registre el intento de pago con estado 'procesando'
- Simule la validación del pago (puede usar un número aleatorio para simular aprobación/rechazo)
- Si el pago es aprobado: actualice el estado del pedido a 'confirmado' y el estado del pago a 'aprobado'
- Si el pago es rechazado: restaure el stock de los productos, actualice el pedido a 'cancelado' y el pago a 'rechazado'
- Retorne TRUE si el pago fue exitoso, FALSE en caso contrario

Tarea 2.3: Crear una función cancelar_pedido que:

- Reciba como parámetro: ID del pedido
- Valide que el pedido pueda ser cancelado (solo si está en estado 'pendiente' o 'confirmado')
- Restaure el stock de todos los productos del pedido
- Actualice el estado del pedido a 'cancelado'

- Registre los movimientos de devolución en el historial de stock
- Retorne TRUE si la cancelación fue exitosa, FALSE en caso

contrario **PISTAS:**

- Use transacciones explícitas (BEGIN/COMMIT/ROLLBACK) en todas las funciones
- Implemente bloques EXCEPTION para capturar y manejar errores
- Considere usar SAVEPOINT para operaciones que puedan requerir reversión parcial
- Use SELECT ... FOR UPDATE para bloquear filas durante validaciones críticas
- Valide todas las condiciones de negocio ANTES de realizar cambios en la base de datos
- Registre todos los movimientos de stock para trazabilidad

PARTE 3: PRUEBAS DE CONCURRENCIA (5 puntos)

Tarea 3.1: Simular escenario de competencia por stock limitado

Abra DOS conexiones simultáneas a la base de datos y ejecute:

Conexión A:

```
sql
-- Configure el nivel de aislamiento apropiado
-- Inicie una transacción
-- Intente crear un pedido para un producto con stock de 1 unidad
-- NO HAGA COMMIT inmediatamente
-- Espere 10 segundos
-- Luego haga COMMIT
```

Conexión B (ejecutar mientras A está en espera):

```
sql
-- Configure el mismo nivel de aislamiento
-- Intente crear un pedido para el MISMO producto
-- Observe qué sucede
```

Documente:

- Qué nivel de aislamiento utilizó y por qué
- Qué sucedió en cada conexión
- Cómo el sistema manejó la situación
- Si hubo bloqueos, cómo se resolvieron

Tarea 3.2: Probar escenario de pago fallido

Ejecute una transacción que:

- Cree un pedido exitosamente (stock reservado)
- Intente procesar el pago
- Fuerce un fallo en el pago
- Verifique que el stock se haya restaurado correctamente

Documente el flujo completo con capturas de los datos antes, durante y después. **Tarea 3.3:** Simular deadlock

Diseñe y ejecute un escenario donde dos transacciones se bloqueen mutuamente (deadlock). Documente:

- Cómo configuró el escenario
- Qué ocurrió cuando se produjo el deadlock
- Cómo PostgreSQL lo resolvió
- Qué estrategias implementaría para evitarlo en producción

PISTAS:

- Para simular competencia, los productos de prueba deben tener stock muy limitado (1-2 unidades)
- Use `pg_sleep()` para crear pausas controladas en las transacciones
- Para forzar un deadlock, dos transacciones deben intentar actualizar las mismas filas en orden inverso
- Documente con capturas de pantalla el estado de las tablas en cada momento
- Consulte las vistas del sistema (`pg_stat_activity`, `pg_locks`) para observar bloqueos activos

PARTE 4: ANÁLISIS Y OPTIMIZACIÓN (3 puntos)

Tarea 4.1: Consultas de monitoreo

Implemente las siguientes consultas y documente sus resultados:

1. Listar todos los pedidos con sus detalles, incluyendo información del cliente y productos
2. Mostrar productos con stock por debajo del mínimo de alerta

3. Generar un reporte de ventas por producto (cantidad vendida, monto total)
4. Listar pedidos cancelados con el motivo de cancelación
5. Mostrar el historial completo de movimientos de stock de un producto específico

Tarea 4.2: Análisis de rendimiento

- Ejecute EXPLAIN ANALYZE en las consultas más complejas
- Identifique oportunidades de optimización
- Proponga y documente al menos 3 mejoras (índices adicionales, reescritura de consultas, etc.)

Tarea 4.3: Gestión de transacciones

Implemente una consulta que muestre:

- Transacciones activas en el sistema
- Bloqueos actuales y su duración
- Queries que están esperando por bloqueos

PISTAS:

- Use JOINS apropiados para relacionar las tablas
- Las funciones de agregación (SUM, COUNT, AVG) serán útiles para los reportes
- Las vistas del sistema pg_stat_activity y pg_locks contienen información sobre transacciones y bloqueos
- EXPLAIN ANALYZE muestra el plan de ejecución real con tiempos
- Los índices compuestos pueden ser útiles para consultas con múltiples filtros

FORMATO DE ENTREGA

Un informe PDF con los pantallazos correspondientes.

Debe entregar UN SOLO archivo SQL (.sql) que contenga:

1. Sección 1: Definición de esquema

- Comando para crear la base de datos
- Comandos CREATE TABLE con todas las restricciones

- Comandos CREATE INDEX para los índices
- Comandos INSERT para los datos de prueba

2. Sección 2: Funciones y procedimientos

- Implementación de crear_pedido
- Implementación de procesar_pago
- Implementación de cancelar_pedido
- Cualquier función auxiliar que haya creado

3. Sección 3: Scripts de prueba

- Scripts utilizados para probar concurrencia
- Scripts para simular escenarios de error
- Scripts para generar deadlocks

4. Sección 4: Consultas de análisis

- Consultas de monitoreo
- Consultas de reportes
- Consultas de rendimiento

5. Sección 5: Documentación (como comentarios SQL)

- Explicación de decisiones de diseño
- Resultados observados en las pruebas de concurrencia
- Análisis de rendimiento y propuestas de mejora
- Conclusiones sobre el comportamiento del sistema

OBSERVACIONES IMPORTANTES

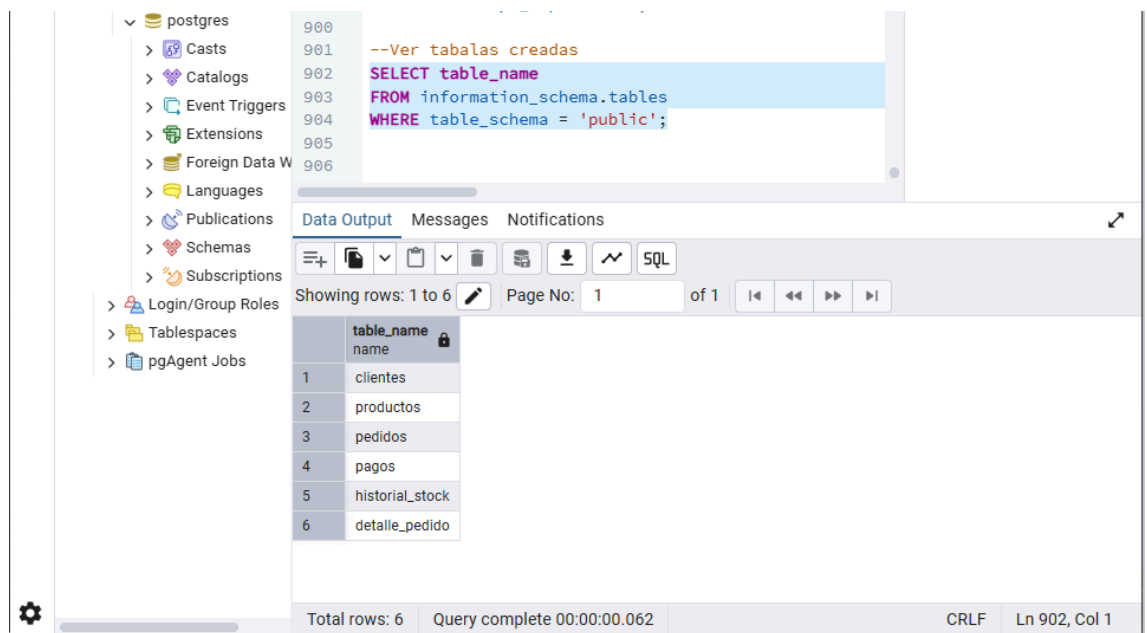
1. El archivo SQL debe poder ejecutarse completamente de forma secuencial sin errores
2. Use comentarios para explicar secciones complejas del código
3. Todos los nombres de objetos (tablas, funciones, columnas) deben seguir una convención consistente
4. Las funciones deben incluir manejo de excepciones y mensajes informativos
5. Documente TODOS los resultados de las pruebas de concurrencia con capturas o descripciones detalladas

DOCUMENTACIÓN

PARTE 1 — CREAR TABLAS Y DATOS

Paso 1: Ver que las tablas están creadas

- ☐ Esta consulta lista todas las tablas que existen en el esquema público de la base de datos.
- ☐ Permite comprobar que se han creado correctamente las tablas requeridas por el sistema (productos, clientes, pedidos, detalle_pedido, pagos y historial_stock).



Paso 2: Ver estructura de una tabla:

- ☐ Esta consulta muestra la estructura de la tabla productos: nombres de columnas, tipo de dato y si pueden aceptar valores nulos.
- ☐ Sirve como evidencia de que la tabla tiene las columnas esperadas, con sus restricciones definidas.

postgres

Casts

Catalogs

Event Triggers

Extensions

Foreign Data W

Languages

Publications

Schemas

Subscriptions

Login/Group Roles

Tablespaces

pgAgent Jobs

905

906

907

908

909

910

911

--Estructura de las tablas

SELECT column_name, data_type, is_nullable

FROM information_schema.columns

WHERE table_name = 'productos';

Data Output

Messages

Notifications

Showing rows: 1 to 8

Page No: 1 of 1

	column_name name	data_type character varying	is_nullable character varying (3)
1	fecha_ultima_actualizacion	timestamp without time zone	YES
2	stock_disponible	integer	NO
3	stock_minimo_alerta	integer	YES
4	codigo_producto	integer	NO
5	precio_unitario	numeric	NO
6	nombre	character varying	NO
7	descripcion	text	YES
8	estado	character varying	YES

Total rows: 8

Query complete 00:00:00.073

CRLF

Ln 907, Col 1

Paso 3: Ver datos insertados:

- ☐ Se muestran los registros insertados en las tablas productos y clientes.
- ☐ Esto permite verificar que existen datos de prueba de al menos 10 productos y 5 clientes y que algunos productos tienen stock bajo (1 o 2 unidades) para realizar pruebas de concurrencia.

857

858

859

860

861

862

863

864

--Datos insertados

SELECT * FROM productos;

SELECT * FROM clientes;

--Crear pedido

SELECT crear_pedido(1, '{"codigo_producto": 1, "cantidad": 3}'::json);

SELECT * FROM pedidos;

SELECT * FROM pedidos ORDER BY id_pedido DESC LIMIT 5;

Data Output

Messages

Notifications

Showing rows: 1 to 10

Page No: 1 of 1

	codigo_producto [PK] integer	nombre character varying (100)	descripcion text	precio_unitario numeric (10,2)	estado character varying (10)	stock_disponible integer	stock_minimo_alerta integer	fecha_ultima_actualizacion timestamp without time zone
1	6	SSD Kingston 1TB NVMe	Unidad SSD M.2 NVMe alta velocidad	420.00	activo	8	2	2025-10-18 20:58:58.123641
2	9	Memoria MicroSD SanDisk 128...	Memoria clase 10 con adaptador	110.00	activo	25	5	2025-10-18 20:58:58.123641
3	2	Mouse Razer DeathAdder	Mouse gamer ergonómico RGB	250.00	activo	12	3	2025-10-18 20:58:58.123641
4	3	Teclado Logitech MX Keys	Teclado inalámbrico retroiluminado	500.00	activo	3	2	2025-10-18 20:58:58.123641
5	7	Impresora Epson EcoTank L3250	Impresora multifuncional Wi-Fi	950.00	activo	5	1	2025-10-18 20:58:58.123641
6	4	Monitor LG UltraGear 27"	Monitor QHD 165Hz para gaming	1850.00	activo	1	1	2025-10-18 20:58:58.123641
7	1	Tablet Samsung Galaxy Tab A8	Tablet 10.5" con 4GB RAM y 64GB almacenamie...	1200.00	activo	0	2	2025-10-18 21:07:58.481697
8	5	Auriculares Sony WH-CH720N	Auriculares inalámbricos con cancelación de ruido	780.00	activo	1	1	2025-10-18 21:09:33.569406
9	10	Webcam Razer Kiyo	Cámara Full HD con luz anular	460.00	activo	2	1	2025-10-18 21:09:33.569406
10	8	Silla Ergonomica T-Dagger	Silla para oficina y juegos	890.00	activo	0	1	2025-10-18 21:13:30.033428

```

911 --Datos insertados
912 SELECT * FROM productos;
913 SELECT * FROM clientes;
914
915 --Crear pedido
916 SELECT crear_pedido(1, '{"codigo_producto": 1, "cantidad": 3}');
917
918

```

	id_cliente [PK] integer	nombre_completo character varying (150)	correo_electronico character varying (100)	telefono character varying (15)	direccion_envio text	fecha_registro timestamp without time zone
1	1	Pedro Salazar	pedrosalazar@example.com	987321654	Av. Primavera 102 - Lima	2025-10-18 20:58:58.123641
2	2	Rosa Martinez	rosamartinez@example.com	991234567	Jr. Las Gardenias 202 - Cusco	2025-10-18 20:58:58.123641
3	3	Miguel Vargas	migueltvargas@example.com	998811223	Av. La Cultura 501 - Arequipa	2025-10-18 20:58:58.123641
4	4	Sofía Castro	sofiacastro@example.com	976123456	Calle Los Tulipanes 303 - Trujillo	2025-10-18 20:58:58.123641
5	5	Daniela Ramos	danielaramos@example.com	954789123	Mz C Lt 7 Urb. San Borja - Lima	2025-10-18 20:58:58.123641

PARTE 2 — FUNCIONES

Paso 4: Crear un pedido

- ☐ La función crear_pedido valida el stock disponible, descuenta la cantidad solicitada y registra el pedido con estado pendiente.

```

915 --Crear pedido
916 SELECT crear_pedido(1, '{"codigo_producto": 1, "can
917
918
919
920

```

	crear_pedido integer
1	5

Paso 5: Ver el pedido creado

Se consulta la tabla de pedidos, detalle_pedido y historial_stock para evidenciar:

- Que se creó el pedido,
- Que se insertaron los detalles,
- Que el stock fue descontado,
- Y que se registró el movimiento en el historial de stock.

```

863 SELECT * FROM pedidos;
864 SELECT * FROM pedidos ORDER BY id_pedido DESC LIMIT 5;
865 SELECT * FROM detalle_pedido WHERE id_pedido = 2;
866 SELECT * FROM historial_stock WHERE id_pedido_relacionado = 2;
867

```

Data Output Messages Notifications							
	id_pedido [PK] integer	id_cliente integer	fecha_pedido timestamp without time zone	estado_pedido character varying (15)	monto_total numeric (10,2)	fecha_ultima_actualizacion timestamp without time zone	
1	3	2	2025-10-18 20:58:58.123641	cancelado	2200.00	2025-10-18 20:58:58.123641	
2	4	2	2025-10-18 20:58:58.123641	pendiente	1850.00	2025-10-18 20:58:58.123641	
3	5	1	2025-10-18 21:00:53.625975	pendiente	3600.00	2025-10-18 21:00:53.625975	
4	7	1	2025-10-18 21:07:58.481697	pendiente	3600.00	2025-10-18 21:07:58.481697	
5	2	1	2025-10-18 20:58:58.123641	cancelado	1240.00	2025-10-18 21:09:33.569406	
6	8	1	2025-10-18 21:13:30.033428	pendiente	890.00	2025-10-18 21:13:30.033428	
7	9	1	2025-10-18 21:13:30.033428	pendiente	890.00	2025-10-18 21:13:30.033428	
8	10	2	2025-10-18 21:13:30.033428	pendiente	890.00	2025-10-18 21:13:30.033428	

```

864 SELECT * FROM pedidos ORDER BY id_pedido DESC LIMIT 5;
865 SELECT * FROM detalle_pedido WHERE id_pedido = 2;
866 SELECT * FROM historial_stock WHERE id_pedido_relacionado = 2;
867

```

Data Output Messages Notifications							
	id_pedido [PK] integer	id_cliente integer	fecha_pedido timestamp without time zone	estado_pedido character varying (15)	monto_total numeric (10,2)	fecha_ultima_actualizacion timestamp without time zone	
1	10	2	2025-10-18 21:13:30.033428	pendiente	890.00	2025-10-18 21:13:30.033428	
2	9	1	2025-10-18 21:13:30.033428	pendiente	890.00	2025-10-18 21:13:30.033428	
3	8	1	2025-10-18 21:13:30.033428	pendiente	890.00	2025-10-18 21:13:30.033428	
4	7	1	2025-10-18 21:07:58.481697	pendiente	3600.00	2025-10-18 21:07:58.481697	
5	5	1	2025-10-18 21:00:53.625975	pendiente	3600.00	2025-10-18 21:00:53.625975	

```

865 SELECT * FROM detalle_pedido WHERE id_pedido = 2;
866 SELECT * FROM historial_stock WHERE id_pedido_relacionado = 2;
867

```

Data Output Messages Notifications							
	id_detalle [PK] integer	id_pedido integer	codigo_producto integer	cantidad_solicitada integer	precio_unitario numeric (10,2)	subtotal numeric (10,2)	cantidad integer
1	2	2	5	1	780.00	780.00	1
2	3	2	10	1	460.00	460.00	1

```

866 SELECT * FROM historial_stock WHERE id_pedido_relacionado = 2;
867

```

Data Output Messages Notifications									
	id_registro [PK] integer	codigo_producto integer	tipo_movimiento character varying (10)	cantidad integer	stock_anterior integer	stock_nuevo integer	id_pedido_relacionado integer	fecha_movimiento timestamp without time zone	usuario_movimiento character varying (100)
1	2	5	salida	1	1	0	2	2025-10-18 20:58:58.123641	postgres
2	3	10	salida	1	2	1	2	2025-10-18 20:58:58.123641	postgres
3	13	5	entrada	1	0	1	2	2025-10-18 21:09:33.569406	postgres
4	14	10	entrada	1	1	2	2	2025-10-18 21:09:33.569406	postgres

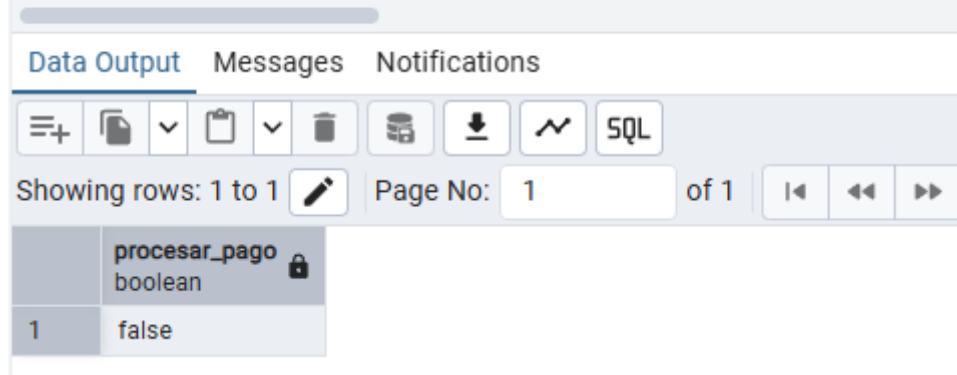
Paso 6: Procesar un pago

- ☐ La función procesar_pago simula la validación de un pago con resultado aprobado o rechazado.
- ☐ Si se aprueba → el estado del pedido cambia a confirmado.
- ☐ Si se rechaza → el pedido se cancela y se restaura el stock.
- ☐ Se debe mostrar el cambio en tablas pedidos, pagos y productos.

```

902  --Procesar pago
903
904  SELECT procesar_pago(1, 'tarjeta', 'REF-001');
905

```



procesar_pago boolean	
1	false

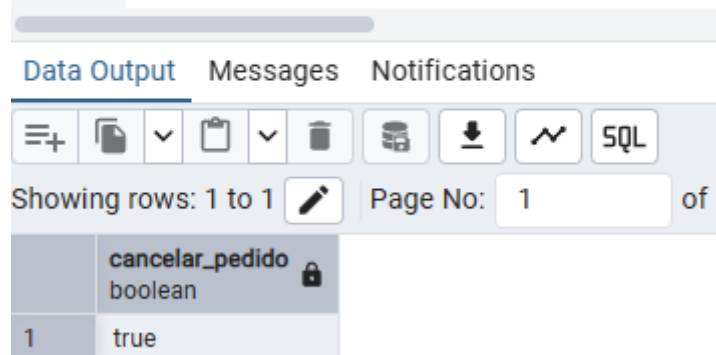
Paso 7: Cancelar un pedido

- ☐ La función cancelar_pedido permite revertir un pedido pendiente o confirmado.
- ☐ Cambia el estado del pedido a cancelado.
- ☐ Restaura el stock de los productos involucrados.
- ☐ Registra el movimiento como “entrada” en historial_stock.
- ☐ Debe evidenciarse que la función devuelve TRUE.

```

891  --Cancelar pedido
892
893  SELECT cancelar_pedido(2);
894
895

```



cancelar_pedido boolean	
1	true

PARTE 3 — CONCURRENCIA Y DEADLOCK

Paso 8: Concurrencia entre dos conexiones

- ☐ Se simula que dos usuarios intentan comprar al mismo tiempo un producto con stock limitado.
- ☐ Gracias al nivel de aislamiento SERIALIZABLE, PostgreSQL evita ventas duplicadas:
- ☐ Una transacción se completa correctamente.
- ☐ La otra queda en espera o falla.
- ☐ Esto demuestra control de concurrencia.

```
618 --Conexion A:
619 BEGIN;
620 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
621 SELECT crear_pedido(1, '{"codigo_producto": 8, "can
622 SELECT pg_sleep(10);
623
624 ROLLBACK;
625
626 --Conexion B:
627 BEGIN;
628 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
629 SELECT crear_pedido(2, '{"codigo_producto": 8, "can
630 COMMIT;
631
632 --ROLLBACK;
633
634 --4.1.
635
636 -- 1. Listar todos los pedidos con sus detalles, inc
637 SELECT
638     p.id_pedido,
639     p.id_cliente,
640     c.nombre_completo AS cliente,
641     pr.codigo_producto,
642     pr.nombre AS producto,
643     dp.cantidad,
644     dp.precio_unitario
```

Data Output Messages Notifications

WARNING: ya hay una transacción en curso
NOTICE: Pedido 10 creado exitosamente con total 890.00
COMMIT

Query returned successfully in 50 msec.

Paso 9: Simular deadlock

- ☐ Se generan dos transacciones que intentan bloquear filas en distinto orden para provocar un deadlock.
- ☐ PostgreSQL detecta el conflicto y aborta automáticamente una de las transacciones para evitar bloqueo permanente.

```

874 --Deadlock
875 --A
876 BEGIN;
877 SELECT * FROM productos WHERE codigo_producto = 8 FOR UPDATE;
878 SELECT pg_sleep(5);
879 SELECT * FROM productos WHERE codigo_producto = 10 FOR UPDATE;
880 --B
881 BEGIN;
882 SELECT * FROM productos WHERE codigo_producto = 10 FOR UPDATE;
883 SELECT pg_sleep(5);
884 SELECT * FROM productos WHERE codigo_producto = 8 FOR UPDATE;

```

PARTE 4 — MONITOREO Y REPORTES

Paso 10: Consultas de reporte

- ☐ Se listan pedidos con detalles, clientes y productos.
- ☐ Permite visualizar el flujo de ventas y validar que la información de la base de datos está relacionada correctamente mediante JOINS.

```

886 --M y R
887 SELECT * FROM pedidos;
888 SELECT * FROM historial_stock;
889

```

Data Output Messages Notifications

SQL

	id_pedido [PK] integer	id_cliente integer	fecha_pedido timestamp without time zone	estado_pedido character varying (15)	monto_total numeric (10,2)	fecha_ultima_actualizacion timestamp without time zone
1	3	2	2025-10-18 20:58:58.123641	cancelado	2200.00	2025-10-18 20:58:58.123641
2	4	2	2025-10-18 20:58:58.123641	pendiente	1850.00	2025-10-18 20:58:58.123641
3	5	1	2025-10-18 21:00:53.625975	pendiente	3600.00	2025-10-18 21:00:53.625975
4	7	1	2025-10-18 21:07:58.481697	pendiente	3600.00	2025-10-18 21:07:58.481697
5	2	1	2025-10-18 20:58:58.123641	cancelado	1240.00	2025-10-18 21:09:33.569406
6	8	1	2025-10-18 21:13:30.033428	pendiente	890.00	2025-10-18 21:13:30.033428
7	9	1	2025-10-18 21:13:30.033428	pendiente	890.00	2025-10-18 21:13:30.033428

Total rows: 8 Query complete 00:00:00.080

```

888 SELECT * FROM historial_stock;
889

```

Data Output Messages Notifications

SQL

	id_registro [PK] integer	codigo_producto integer	tipo_movimiento character varying (10)	cantidad integer	stock_anterior integer	stock_nuevo integer	id_pedido_relacionado integer	fecha_movimiento timestamp without time zone	usuario_movimiento character varying (100)
1	2	5	salida	1	1	0	2	2025-10-18 20:58:58.123641	postgres
2	3	10	salida	1	2	1	2	2025-10-18 20:58:58.123641	postgres
3	4	2	salida	1	12	11	3	2025-10-18 20:58:58.123641	postgres
4	5	3	salida	2	4	2	3	2025-10-18 20:58:58.123641	postgres
5	6	7	salida	1	5	4	3	2025-10-18 20:58:58.123641	postgres
6	7	2	entrada	1	11	12	3	2025-10-18 20:58:58.123641	postgres
7	8	3	entrada	1	2	3	3	2025-10-18 20:58:58.123641	postgres

Total rows: 16 Query complete 00:00:00.068

Showing rows: 1 to 16 Page No: 1

Successfully run. Total query runtime

Paso 11: Consultas EXPLAIN ANALYZE

La instrucción EXPLAIN ANALYZE muestra el plan de ejecución de una consulta.

```
719 --Explain
720 EXPLAIN ANALYZE
721 SELECT
722     p.id_pedido,
723     c.nombre_completo AS cliente,
```

Data Output Messages Notifications

SQL

QUERY PLAN	
text	
1	Merge Join (cost=27.21..27.28 rows=6 width=600) (actual time=0.120..0.124 rows=11 loops=1)
2	Merge Cond: (p.id_pedido = dp.id_pedido)
3	-> Sort (cost=12.77..12.78 rows=3 width=346) (actual time=0.084..0.085 rows=8 loops=1)
4	Sort Key: p.id_pedido
5	Sort Method: quicksort Memory: 25kB
6	-> Hash Join (cost=1.07..12.75 rows=3 width=346) (actual time=0.075..0.077 rows=8 loops=1)
7	Hash Cond: (c.id_cliente = p.id_cliente)
8	-> Seq Scan on clientes c (cost=0.00..11.20 rows=120 width=322) (actual time=0.043..0.043 rows=5 loo...
9	-> Hash (cost=1.03..1.03 rows=3 width=32) (actual time=0.013..0.014 rows=8 loops=1)
10	Buckets: 1024 Batches: 1 Memory Usage: 9kB
11	-> Seq Scan on pedidos p (cost=0.00..1.03 rows=3 width=32) (actual time=0.009..0.011 rows=8 loo...
12	-> Sort (cost=14.44..14.45 rows=6 width=258) (actual time=0.033..0.034 rows=11 loops=1)
13	Sort Key: dp.id_pedido
14	Sort Method: quicksort Memory: 25kB
15	-> Hash Join (cost=1.14..14.36 rows=6 width=258) (actual time=0.027..0.030 rows=11 loops=1)
16	Hash Cond: (pr.codigo_producto = dp.codigo_producto)
17	-> Seq Scan on productos pr (cost=0.00..12.30 rows=230 width=222) (actual time=0.005..0.006 rows=...
18	-> Hash (cost=1.06..1.06 rows=6 width=44) (actual time=0.009..0.009 rows=11 loops=1)
19	Buckets: 1024 Batches: 1 Memory Usage: 9kB
20	-> Seq Scan on detalle_pedido dp (cost=0.00..1.06 rows=6 width=44) (actual time=0.005..0.006 row...
21	Planning Time: 0.785 ms
22	Execution Time: 0.175 ms

Servers (1)

- PostgreSQL 17
 - Databases (9)
 - BS2
 - Parcial
 - Practica 4
 - Practica 5
 - Trabajo 3
 - ecommerce_lab
 - Casts
 - Catalogs
 - Event Triggers
 - Extensions
 - Foreign Data W
 - Languages
 - Publications
 - Schemas
 - Subscriptions
 - laboratorio_indice
 - laboratorio_optim
 - postgres
 - Login/Group Roles
 - Tablespaces
 - pgAgent Jobs

ecommerce_lab/postgres@PostgreSQL 17

Query

Query History

Scratch Pad x

740

741

742

743

744

745

746

747

748

SUM(dp.cantidad) AS total_vendido,

SUM(dp.subtotal) AS monto_total

FROM detalle_pedido dp

JOIN pedidos p ON dp.id_pedido = p.id_pedido

JOIN productos pr ON dp.codigo_producto = pr.codigo_

WHERE p.estado_pedido IN ('confirmado', 'enviado')

GROUP BY pr.codigo_producto, pr.nombre

ORDER BY monto_total DESC;

Data Output

Messages

Notifications

Showing rows: 1 to 23

Page No: 1 of 1

SQL

QUERY PLAN

text

1 Sort (cost=15.57..15.58 rows=4 width=262) (actual time=0.050..0.051 rows=0 loops=1)

2 Sort Key: (sum(dp.subtotal)) DESC

3 Sort Method: quicksort Memory: 25kB

4 -> GroupAggregate (cost=15.44..15.53 rows=4 width=262) (actual time=0.044..0.045 rows=0 loops=1)

5 Group Key: pr.codigo_producto

6 -> Sort (cost=15.44..15.45 rows=4 width=242) (actual time=0.043..0.044 rows=0 loops=1)

7 Sort Key: pr.codigo_producto

8 Sort Method: quicksort Memory: 25kB

9 -> Hash Join (cost=2.20..15.40 rows=4 width=242) (actual time=0.029..0.030 rows=0 loops=1)

10 Hash Cond: (pr.codigo_producto = dp.codigo_producto)

11 -> Seq Scan on productos pr (cost=0.00..12.30 rows=230 width=222) (actual time=0.010..0.010 ro...

12 -> Hash (cost=2.15..2.15 rows=4 width=24) (actual time=0.015..0.016 rows=0 loops=1)

13 Buckets: 1024 Batches: 1 Memory Usage: 8kB

14 -> Hash Join (cost=1.06..2.15 rows=4 width=24) (actual time=0.015..0.015 rows=0 loops=1)

15 Hash Cond: (dp.id_pedido = p.id_pedido)

16 -> Seq Scan on detalle_pedido dp (cost=0.00..1.06 rows=6 width=28) (actual time=0.006..0.0...

17 -> Hash (cost=1.04..1.04 rows=2 width=4) (actual time=0.006..0.007 rows=0 loops=1)

18 Buckets: 1024 Batches: 1 Memory Usage: 8kB

19 -> Seq Scan on pedidos p (cost=0.00..1.04 rows=2 width=4) (actual time=0.006..0.006 r...

20 Filter: ((estado_pedido)::text = ANY ('{confirmado,enviado}':text[]))

21 Rows Removed by Filter: 8

Total rows: 23

Query complete 00:00:00.063

CRLF

Ln 747, Col 27

```

749 EXPLAIN ANALYZE
750 SELECT
751     h.codigo_producto,
752     pr.nombre AS producto,
753     h.tipo_movimiento,
754     h.cantidad,
755     h.fecha_movimiento
756 FROM historial_stock h
757 JOIN productos pr ON h.codigo_producto = pr.codigo_p
758 ORDER BY h.fecha_movimiento DESC;
759
760 ----Identificar oportunidades de optimización
761
762 --No hay índice útil.
763 --Sort costoso

```

Data Output Messages Notifications

Showing rows: 1 to 11 Page No: 1 of 1

	QUERY PLAN	
	text	
1	Sort (cost=14.60..14.62 rows=9 width=272) (actual time=0.037..0.038 rows=16 loops=1)	
2	Sort Key: h.fecha_movimiento DESC	
3	Sort Method: quicksort Memory: 26kB	
4	-> Hash Join (cost=1.20..14.46 rows=9 width=272) (actual time=0.026..0.030 rows=16 loops=1)	
5	Hash Cond: (pr.codigo_producto = h.codigo_producto)	
6	-> Seq Scan on productos pr (cost=0.00..12.30 rows=230 width=222) (actual time=0.011..0.011 rows=10 ...)	
7	-> Hash (cost=1.09..1.09 rows=9 width=54) (actual time=0.010..0.011 rows=16 loops=1)	
8	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
9	-> Seq Scan on historial_stock h (cost=0.00..1.09 rows=9 width=54) (actual time=0.005..0.007 rows=1...)	
10	Planning Time: 0.160 ms	
11	Execution Time: 0.056 ms	

Paso 12: Consultar transacciones y bloqueos

- ☐ Las consultas a pg_stat_activity y pg_locks muestran:
- ☐ Transacciones activas,
- ☐ Sesiones bloqueadas,
- ☐ Bloqueos vigentes.
- ☐ Esto permite monitorear el estado del sistema en tiempo real y comprobar la concurrencia durante las pruebas.

891--Transaccion y bloqueos

892SELECT * FROM pg_stat_activity;

893SELECT * FROM pg_locks;

Data OutputMessagesNotifications

Showing rows: 1 to 11Page No: 1of 1

datid	dbname	pid	leader_pid	usesysid	username	application_name	client_addr	client_hostname	client_port	backend_start	xact_start	query_start	
1	5	postgres	14320	[null]	10	postgres	pgAdmin 4 - DB:postgres	:::1	[null]	26283	2025-10-18 18:16:20.371488-05	[null]	2025-10-18 20:57
2	32826	Practica 5	8112	[null]	10	postgres	pgAdmin 4 - DB:Practica 5	:::1	[null]	26297	2025-10-18 18:16:20.766276-05	[null]	2025-10-18 18:16
3	40969	Parcial	3644	[null]	10	postgres	pgAdmin 4 - DB:Parcial	:::1	[null]	26328	2025-10-18 18:16:35.420321-05	[null]	2025-10-18 20:57
4	16640	BS2	20340	[null]	10	postgres	pgAdmin 4 - DB:BS2	:::1	[null]	58299	2025-10-18 20:57:20.655573-05	[null]	2025-10-18 20:57
5	41102	ecommerce_lab	15344	[null]	10	postgres	pgAdmin 4 - DB:ecommerce_lab	:::1	[null]	58312	2025-10-18 20:57:30.045287-05	[null]	2025-10-18 20:57
6	41102	ecommerce_lab	21636	[null]	10	postgres	pgAdmin 4 - CONN:4621825	:::1	[null]	58336	2025-10-18 20:57:42.196094-05	2025-10-18 21:33:00.073058-05	2025-10-18 21:42
7	[null]	[null]	15820	[null]	[null]	[null]	[null]	[null]	[null]	[null]	2025-10-18 18:06:29.503137-05	[null]	[null]
8	[null]	[null]	13940	[null]	10	postgres	[null]	[null]	[null]	[null]	2025-10-18 18:06:29.504533-05	[null]	[null]
9	[null]	[null]	8712	[null]	[null]	[null]	[null]	[null]	[null]	[null]	2025-10-18 18:06:29.501568-05	[null]	[null]
10	[null]	[null]	19748	[null]	[null]	[null]	[null]	[null]	[null]	[null]	2025-10-18 18:06:29.578721-05	[null]	[null]
11	[null]	[null]	18500	[null]	[null]	[null]	[null]	[null]	[null]	[null]	2025-10-18 18:06:29.578725-05	[null]	[null]

Total rows: 11Query complete 00:00:00.075

894SELECT * FROM pg_stat_activity;

895SELECT * FROM pg_locks;

896

Data OutputMessagesNotifications

Showing rows: 1 to 69Page No: 1of 1

locktype	database	relation	page	tuple	virtual	transaction	class	objid	objsubid	virtual	pid	mode	granted	fastpath	waitstart
text	oid	oid	integer	smallint	text	xid	oid	oid	smallint	text	integer	text	boolean	boolean	timestamp with time zone
1	relation	41102	3455	[null]	[null]	[null]	[null]	[null]	[null]	9/84	21636	AccessShareLock	true	true	[null]
2	relation	41102	2663	[null]	[null]	[null]	[null]	[null]	[null]	9/84	21636	AccessShareLock	true	true	[null]
3	relation	41102	2662	[null]	[null]	[null]	[null]	[null]	[null]	9/84	21636	AccessShareLock	true	true	[null]
4	relation	41102	2704	[null]	[null]	[null]	[null]	[null]	[null]	9/84	21636	AccessShareLock	true	true	[null]
5	relation	41102	2703	[null]	[null]	[null]	[null]	[null]	[null]	9/84	21636	AccessShareLock	true	true	[null]
6	relation	41102	2659	[null]	[null]	[null]	[null]	[null]	[null]	9/84	21636	AccessShareLock	true	true	[null]
7	relation	41102	2658	[null]	[null]	[null]	[null]	[null]	[null]	9/84	21636	AccessShareLock	true	true	[null]
8	relation	41102	2615	[null]	[null]	[null]	[null]	[null]	[null]	9/84	21636	AccessShareLock	true	true	[null]
9	relation	41102	1259	[null]	[null]	[null]	[null]	[null]	[null]	9/84	21636	AccessShareLock	true	true	[null]
10	relation	41102	1247	[null]	[null]	[null]	[null]	[null]	[null]	9/84	21636	AccessShareLock	true	true	[null]
11	relation	41102	1249	[null]	[null]	[null]	[null]	[null]	[null]	9/84	21636	AccessShareLock	true	true	[null]
12	relation	41102	41121	[null]	[null]	[null]	[null]	[null]	[null]	9/84	21636	RowShareLock	true	true	[null]
13	relation	41102	41120	[null]	[null]	[null]	[null]	[null]	[null]	9/84	21636	RowShareLock	true	true	[null]
14	relation	41102	41119	[null]	[null]	[null]	[null]	[null]	[null]	9/84	21636	RowShareLock	true	true	[null]

Total rows: 69Query complete 00:00:00.084

Successfully run. Total query runtime: 84 msec. 69 rows affected. X

CRFLLn 892, Col 1