

Trabajo Práctico Integrador

Bases de Datos I - Entidades Pedido-Producto

❖ **Alumnos:**

- Sussini Guanziroli, Patricio
- Vazquez, Matías Ezequiel
- Zárate, Lucas Martín

❖ **Materia:** Bases de datos I

❖ **Tutora:** Constanza Uño

❖ **Profesora:** Cinthia Rigoni

❖ **Fecha de Presentación:** 23/10/2025

Trabajo Práctico Integrador.....	1
Bases de Datos I - Entidades Pedido-Producto.....	1
Diseño y Selección de Tópicos.....	4
Etapa 1.....	5
Detalle del diseño de la base de datos:.....	5
Inserts:.....	5
Etapa 2:.....	7
Descripción del Mecanismo de Carga de Datos.....	7
DISTRIBUCIÓN DE REGISTROS.....	7
TABLAS SEMILLA Y MAESTRAS.....	7
TÉCNICA DE GENERACIÓN DE DATOS.....	8
INTEGRIDAD REFERENCIAL.....	8
DISTRIBUCIONES Y VALORES.....	10
Explicación sobre verificaciones realizadas:.....	11
VERIFICACIÓN 1: Conteo Total de Registros.....	11
VERIFICACIÓN 2: Detección de Foreign Keys Huérfanas.....	12
VERIFICACIÓN 3: Cardinalidad - Pedidos sin Productos.....	13
VERIFICACIÓN 4: Distribución de Productos por Pedido.....	14
VERIFICACIÓN 5: Cardinalidad - Envíos con Estado Inválido.....	14
VERIFICACIÓN 6: Rangos de Dominio - Precios Válidos.....	15
VERIFICACIÓN 7: Integridad Matemática - Totales de Pedidos.....	15
VERIFICACIÓN 8: Integridad Temporal - Fechas Coherentes de Envío.....	16
Etapa 3:.....	17
Diseño de Consultas con Utilidad Realista.....	17
Consulta 1: Top 5 productos más vendidos.....	17
Consulta 2: Clientes Fieles (con +10 pedidos).....	18
Consulta 3: Top 5 Clientes con más dinero invertido en la tienda.....	19
Consulta 4: Provincias con Ventas > \$100k y su producto estrella.....	20
Creación de la Vista.....	22
Elección de vista:.....	22
Creación de Índices y Mediciones de Tiempos.....	24
Conclusión del estudio:.....	29
Explain de las consultas:.....	30
Consulta 2:.....	30
Consulta 3:.....	31
Etapa 4:.....	32
Creación de usuario con permisos acotados (principio de mínimos privilegios).....	32
Puntos a mejorar en un caso real:.....	33
Vistas que ocultan información sensible.....	34
Puntos a tener en cuenta o de mejora.....	35
Concepto importante:.....	35
Violacion de restricciones (PK, FK, UNIQUE, CHECK).....	36
Violacion de PK (duplicado de llave primaria).....	36
Violacion de FK (referencia a cliente inexistente).....	37

Violacion de UNIQUE (código de producto duplicado).....	38
Violacion de CHECK (precio negativo, cantidad inválida).....	38
Implementar una consulta segura (JAVA con PreparedStatement) + prueba anti-SQL injection.....	39
Etapa 5.....	41
Simulación de deadlock.....	41
READ COMMITTED.....	48
REPEATABLE READ.....	55
Anexo.....	62
Uso de IA por etapa:.....	62
Etapa 1:.....	62
Etapa 2:.....	62
Etapa 3:.....	62
Referencias.....	62

Diseño y Selección de Tópicos

Modelado entidades Pedido - Envío.

Para el diseño de la base de datos para el Trabajo Integrador seleccionamos el tema de Pedido-Productos y sus relaciones.

Dichas entidades en el contexto de un diseño de base de datos constituye una relación de N:M (muchos a muchos).

- Un solo **Pedido** puede contener varios **Producto/s**. Por ejemplo: Una compra en línea en una tienda electrónica puede contener un mouse, dos discos sólidos y un gabinete, constituyendo un solo **pedido**.
- Un solo **Producto**, con un `id_producto` único puede estar incluído en varios **pedidos**.
- Cada pedido luego constituye una existencia de un envío.

Nuestro proyecto simula una base de datos de una tienda online con un extenso catálogo de productos de diferentes categorías que satisfacen una amplia gama de necesidades para todos los públicos y usos.

La tienda se llama “*Punto de Ventas Argentino*” y su manejo es responsabilidad de una base de datos relacional en SQL.

A continuación veremos el proceso de diseño, carga y consulta de la base de datos del Punto de Ventas así como su información en materia de Seguridad e Integridad de datos y su control de Concurrencia y Transacciones.

Etapa 1

Detalle del diseño de la base de datos:

[Ver 01_esquemas.sql](#)

Inserts:

Vamos probando y mostramos 2 inserts correctos y dos que fallan

Inserts con logs de errores por los siguientes motivos:

1.

```

1. 20:30:07 USE tp_i_pedido_envio;
2. 20:30:13 INSERT INTO LOCALIDADES (ciudad, provincia, codigo_postal) VALUES ('Buenos Aires', 'Buenos Aires', '1406');
3. 20:30:13 INSERT INTO LOCALIDADES (ciudad, provincia, codigo_postal) VALUES ('Córdoba', 'Córdoba', '5000');
4. 20:30:13 INSERT INTO LOCALIDADES (ciudad, provincia, codigo_postal) VALUES ('Santa Fe', 'Santa Fe', '2000')
5. 20:30:13 INSERT INTO CLIENTE (cliente_nombre, cliente_email, cliente_telefono, direccion_entrega, id_localidad)
VALUES ('Ramon Juarez', 'ramon.juarez_email.com', '11-8231-7620', 'Av. Siempre Viva 14', 2); -- ERROR CHECK '@'
11

```

Action Output

Time	Action	Message	Duration / Fetch
1	1 20:30:07 USE tp_i_pedido_envio;	0 rows affected	0.000 sec
2	2 20:30:13 INSERT INTO LOCALIDADES (ciudad, provincia, codigo_postal) VALUES ('Buenos Aires', 'Buenos Aires', '1406');	3 rows affected	0.016 sec
3	3 20:30:13 INSERT INTO LOCALIDADES (ciudad, provincia, codigo_postal) VALUES ('Córdoba', 'Córdoba', '5000');	3 rows affected	0.016 sec
4	4 20:30:13 INSERT INTO LOCALIDADES (ciudad, provincia, codigo_postal) VALUES ('Santa Fe', 'Santa Fe', '2000');	3 rows affected	0.016 sec
5	5. 20:30:13 INSERT INTO CLIENTE (cliente_nombre, cliente_email, cliente_telefono, direccion_entrega, id_localidad)	0 rows affected	0.000 sec
	VALUES ('Ramon Juarez', 'ramon.juarez_email.com', '11-8231-7620', 'Av. Siempre Viva 14');	Error Code: 3115. Check constraint 'uk_cliente_email' is violated.	0.016 sec

2.

```

17
18. • INSERT INTO PEDIDO_PRODUCTO (id_pedido, id_producto, cantidad, precio_unitario, subtotal)
19.   VALUES (1, 3, 1, 450000.00, 450000.00), (2, 2, 2, 9000.00, 18000.00); -- ERROR VALIDACION SUBTOTAL - (PRECIO_UNITARIO * CANTIDAD) DEBE SER 0
20

```

Action Output

Time	Action	Message	Duration / Fetch
1	1 20:30:07 USE tp_i_pedido_envio;	0 rows affected	0.000 sec
2	2 20:30:13 INSERT INTO LOCALIDADES (ciudad, provincia, codigo_postal) VALUES ('Buenos Aires', 'Buenos Aires', '1406');	3 rows affected	0.016 sec
3	3 20:30:13 INSERT INTO CLIENTE (cliente_nombre, cliente_email, cliente_telefono, direccion_entrega, id_localidad)	3 rows affected	0.016 sec
4	4. 20:30:13 INSERT INTO CLIENTE (cliente_nombre, cliente_email, cliente_telefono, direccion_entrega, id_localidad)	0 rows affected	0.000 sec
5	5 20:30:13 INSERT INTO PEDIDO (id_pedido, fecha_pedido, estado_pedido, total_observaciones)	1 rows affected	0.000 sec
6	6. 20:30:30 INSERT INTO PEDIDO (id_pedido, fecha_pedido, estado_pedido, total_observaciones)	1 rows affected	0.000 sec
7	7 20:30:30 INSERT INTO PEDIDO_PRODUCTO (id_pedido, id_producto, cantidad_precio_unidad, subtotal)	2 rows affected	0.000 sec
	VALUES (1, 1, 450000.00, 450000.00), (2, 2, 9000.00, 18000.00);	Error Code: 3115. Check constraint 'tp_i_pedido_envio_ibfk_1' is violated.	0.016 sec
	20:30:30 INSERT INTO PEDIDO_PRODUCTO (id_pedido, id_producto, cantidad_precio_unidad, subtotal)	0 rows affected	0.000 sec
	VALUES (1, 1, 450000.00, 450000.00), (2, 2, 9000.00, 18000.00);	Error Code: 3115. Check constraint 'tp_i_pedido_envio_ibfk_1' is violated.	0.016 sec

Inserts correctos por los siguientes motivos:

1.

```

1. * USE tp_i_pedido_envio;
2.
3. • INSERT INTO LOCALIDADES (ciudad, provincia, codigo_postal)
4.   VALUES ('Buenos Aires', 'Buenos Aires', '1406'), ('Córdoba', 'Córdoba', '5000'), ('Rosario', 'Santa Fe', '2000');
5.

```

Action Output

Time	Action	Message	Duration / Fetch
1	1 20:30:07 USE tp_i_pedido_envio;	0 rows affected	0.000 sec
2	2 20:30:13 (INSERT INTO LOCALIDADES (ciudad, provincia, codigo_postal) VALUES ('Buenos Aires', 'Buenos Aires', '1406'))	3 rows affected	0.016 sec

2.

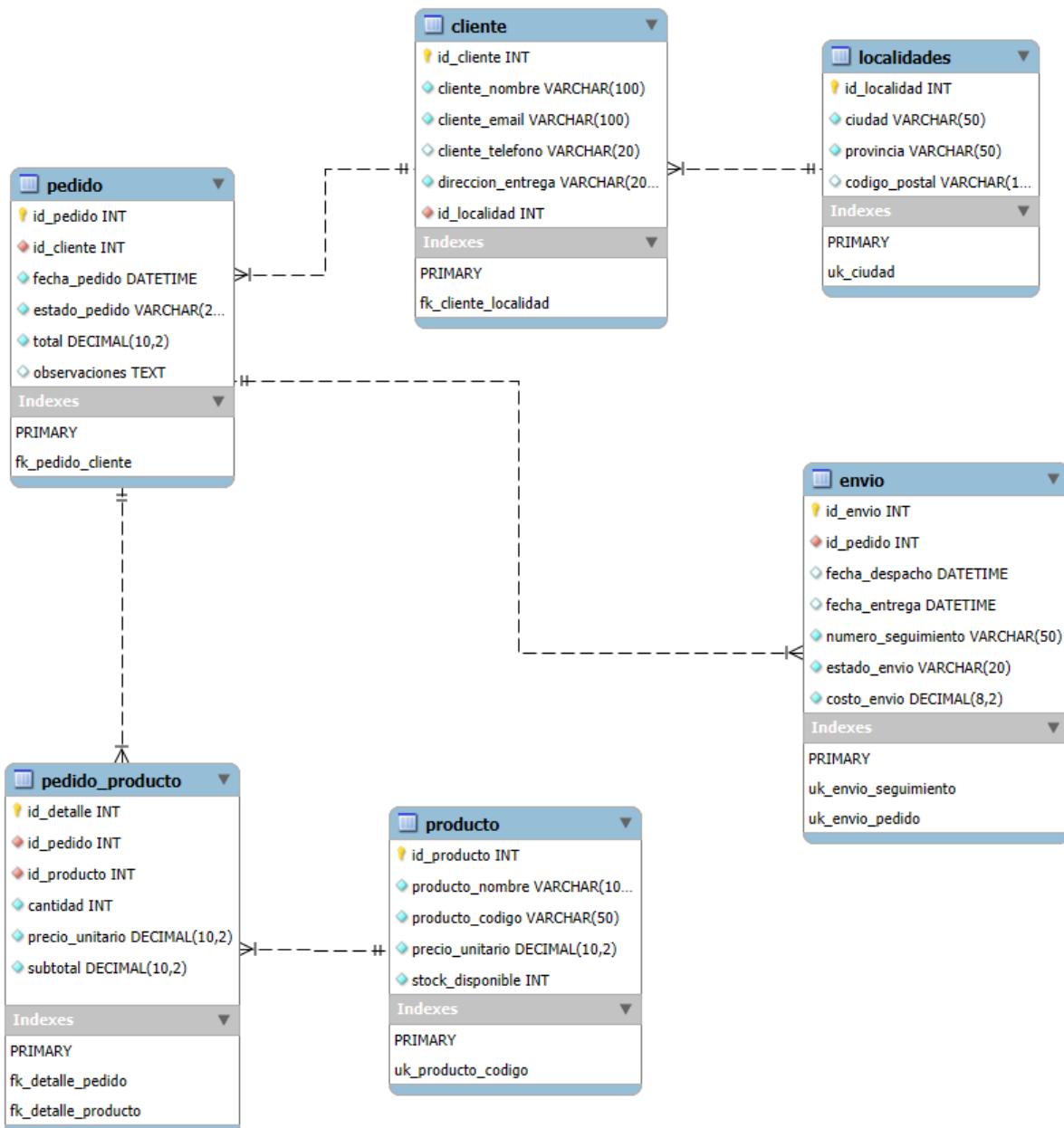
```

5
6. • INSERT INTO CLIENTE (cliente_nombre, cliente_email, cliente_telefono, direccion_entrega, id_localidad)
7.   VALUES ('Juan Pérez', 'juan.perez@email.com', '11-4567-8900', 'Av. Rivadavia 1234', 1);
8

```

Action Output

Time	Action	Message	Duration / Fetch
1	1 20:30:07 USE tp_i_pedido_envio;	0 rows affected	0.000 sec
2	2 20:30:13 (INSERT INTO LOCALIDADES (ciudad, provincia, codigo_postal) VALUES ('Buenos Aires', 'Buenos Aires', '1406'))	3 rows affected	0.016 sec
3	3 20:30:13 (INSERT INTO CLIENTE (cliente_nombre, cliente_email, cliente_telefono, direccion_entrega, id_localidad))	1 rows affected	0.016 sec



Referencias: Anexo Ref 0.1.

Etapa 2:

Descripción del Mecanismo de Carga de Datos

Para la Etapa 2 del trabajo, generamos aproximadamente 235.000 registros en total, distribuidos entre las 6 tablas principales del sistema. El objetivo fue crear un volumen suficiente de datos para poder hacer pruebas de rendimiento realistas en las etapas siguientes.

DISTRIBUCIÓN DE REGISTROS

La cantidad de registros por tabla resultó de la siguiente manera:

- LOCALIDADES: 48 registros
- CLIENTE: 30.000 registros
- PRODUCTO: 5.000 registros
- PEDIDO: 100.000 registros
- PEDIDO_PRODUCTO: 200.000 registros (consideramos aproximadamente 2 productos por pedido)
- ENVIO: 33.000 registros (en donde solo los pedidos despachados o entregados fueron considerados)

TABLAS SEMILLA Y MAESTRAS

Para generar los datos utilizamos dos tablas base:

- LOCALIDADES

Esta tabla la cargamos manualmente con 48 localidades reales de Argentina, incluyendo ciudad, provincia y código postal, la misma fue utilizada como referencia para asignar ubicaciones a los clientes y que los datos tengan sentido geográficamente.

- CATEGORÍAS (tabla temporal)

Creamos una tabla temporal con 10 categorías de productos (Electrónica, Hogar, Deportes, Libros, Juguetes, Ropa, Alimentos, Herramientas, Jardín y Oficina). Esto nos permitió generar nombres de productos más descriptivos y variados en el universo de datos final.

TÉCNICA DE GENERACIÓN DE DATOS

Para generar grandes cantidades de registros usamos la técnica de "producto cartesiano controlado", en donde creamos subconsultas que generan números del 0 al 9 y las combinamos con CROSS JOIN para obtener secuencias numéricas largas:

- Para 30.000 números: combinamos 5 subconsultas de dígitos (0-9)
- Para 100.000 números: igual que el ítem anterior, pero extendiendo el límite
- Para 5.000 números: utilizamos 4 subconsultas con límite

Esta técnica nos permitió evitar usar WITH RECURSIVE y mantener la compatibilidad.

Una vez que obtuvimos los números, los utilizamos para construir los datos mediante funciones como CONCAT para nombres y emails, LPAD para códigos con ceros a la izquierda, y el operador módulo (%) para distribuir valores.

INTEGRIDAD REFERENCIAL

A fin de validar que todas las Foreign Keys apuntan a Primary Keys válidas, aplicamos las siguientes estrategias:

- CLIENTE → LOCALIDADES

Utilizamos el operador módulo para distribuir los clientes entre las 48 localidades:

$$id_localidad = (numero_cliente \% 48) + 1$$

De esta forma nos aseguramos de que todos los valores queden entre 1 y 48.

- PEDIDO → CLIENTE

Asignamos los clientes de forma aleatoria dentro del rango válido:

$$id_cliente = FLOOR(1 + RAND() * 30000)$$

Garantizando así valores entre 1 y 30.000.

- PEDIDO_PRODUCTO → PEDIDO y PRODUCTO

Utilizamos una fórmula para asegurar que los productos se distribuyan de forma variada pero controlada:

$$id_producto = ((id_pedido * 17) \% 5000) + 1$$

Al hacer el JOIN directo con la tabla PRODUCTO, nos aseguramos de que el id exista.

- ENVIO → PEDIDO

Los envíos solo los generamos para aquellos pedidos que estén en estado 'Despachado' o 'Entregado':

WHERE p.estado_pedido IN ('Despachado', 'Entregado')

Además, el constraint UNIQUE en id_pedido asegura que cada pedido tenga como máximo un envío.

CARDINALIDADES

Determinamos las relaciones entre las tablas de la siguiente manera:

- CLIENTE : PEDIDO (1:N)

Cada cliente puede tener varios pedidos. La distribución de pedidos la realizamos con RAND() para que algunos clientes tengan más pedidos que otros.

- PEDIDO : PEDIDO_PRODUCTO (1:N)

Todos los pedidos tienen al menos 1 producto:

- El 70% de los pedidos tienen 2 productos (usando *WHERE id_pedido % 10 < 7*)
- El 30% de los pedidos tienen 3 productos (usando *WHERE id_pedido % 10 < 3*)

- PRODUCTO : PEDIDO_PRODUCTO (1:N)

Cada producto puede estar en muchos pedidos diferentes.

- PEDIDO : ENVIO (1:1)

Solo los pedidos que están despachados o entregados tienen un envío asociado.

DISTRIBUCIONES Y VALORES

Para que los datos sean más realistas, aplicamos las siguientes distribuciones:

- Estados de PEDIDO: Utilizamos CASE con RAND() para distribuir los estados aproximadamente en:
 - 15% Pendiente
 - 20% Confirmado
 - 20% En Preparación
 - 20% Despachado
 - 17% Entregado
 - 8% Cancelado
- Fechas: Los pedidos los distribuimos a lo largo del año usando DATE_SUB con un intervalo aleatorio entre 0 y 365 días.
- Precios: Los productos tienen precios entre \$500 y \$50.000, generados con:
 - $ROUND(500 + RAND() * 49500, 2)$
- Cantidades: Las cantidades por cada línea de pedido van de 1 a 5 unidades, usando el operador módulo.

Explicación sobre verificaciones realizadas:

VERIFICACIÓN 1: Conteo Total de Registros

Para esta primera verificación, decidimos realizar un conteo completo de todos los registros en cada tabla de la base de datos. Utilizando una consulta con UNION ALL para combinar los resultados de cada tabla en una sola salida, lo que nos permite visualizar de forma clara la distribución de datos.

Al final de la consulta, agregamos una fila que suma el total general de registros. Esto nos sirve para confirmar que alcanzamos el volumen de datos que nos propusimos cargar. La consulta recorre cada tabla (LOCALIDADES, CLIENTE, PRODUCTO, PEDIDO, PEDIDO_PRODUCTO, ENVIO) y cuenta cuántos registros hay en cada una.

Este conteo nos permite documentar el tamaño del conjunto de datos y verificar que la distribución entre las tablas sea proporcional a lo planificado.

```
SELECT 'LOCALIDADES' AS Tabla, COUNT(*) AS Registros FROM
LOCALIDADES
UNION ALL
SELECT 'CLIENTE', COUNT(*) FROM CLIENTE
UNION ALL
SELECT 'PRODUCTO', COUNT(*) FROM PRODUCTO
UNION ALL
SELECT 'PEDIDO', COUNT(*) FROM PEDIDO
UNION ALL
SELECT 'PEDIDO_PRODUCTO', COUNT(*) FROM PEDIDO_PRODUCTO
UNION ALL
SELECT 'ENVIO', COUNT(*) FROM ENVIO
UNION ALL
SELECT '==== TOTAL ===',
(SELECT SUM(cnt) FROM (
    SELECT COUNT(*) AS cnt FROM LOCALIDADES
    UNION ALL SELECT COUNT(*) FROM CLIENTE
    UNION ALL SELECT COUNT(*) FROM PRODUCTO
    UNION ALL SELECT COUNT(*) FROM PEDIDO
    UNION ALL SELECT COUNT(*) FROM PEDIDO_PRODUCTO
    UNION ALL SELECT COUNT(*) FROM ENVIO
) t);
```

VERIFICACIÓN 2: Detección de Foreign Keys Huérfanas

En esta verificación buscamos identificar si existen registros en las tablas "hijas" que refieren a registros inexistentes en las tablas "padre". Esto se conoce como validando así si encontramos alguna foreign key huérfana, lo cual representaría una violación de la integridad referencial.

Para detectar estos casos, utilizamos LEFT JOIN combinado con WHERE / IS NULL, al momento de realizar un LEFT JOIN entre la tabla hija y la tabla padre, si un registro de la tabla hija no encuentra su correspondiente en la tabla padre, el resultado del JOIN tendrá NULL en las columnas de la tabla padre.

Entonces, al filtrar con WHERE l.id_localidad IS NULL, estamos seleccionando exactamente esos casos problemáticos.

Aplicamos la misma lógica a todas las relaciones del modelo:

- Clientes que refieren localidades inexistentes
- Pedidos que refieren clientes inexistentes
- Detalles de pedido que refieren pedidos o productos inexistentes
- Envíos que refieren pedidos inexistentes

El resultado esperado para todas las consultas es 0, ya que si se encuentra, aunque sea un registro huérfano, significa que mi proceso de generación de datos tiene un error a corregir.

```
SELECT 'CLIENTE → LOCALIDADES' AS Relación, COUNT(*) AS FK_Huérfanas
FROM CLIENTE c
LEFT JOIN LOCALIDADES l ON c.id_localidad = l.id_localidad
WHERE l.id_localidad IS NULL
```

```
UNION ALL
```

```
SELECT 'PEDIDO → CLIENTE', COUNT(*)
FROM PEDIDO p
LEFT JOIN CLIENTE c ON p.id_cliente = c.id_cliente
WHERE c.id_cliente IS NULL
```

```
UNION ALL
```

```
SELECT 'PEDIDO_PRODUCTO → PEDIDO', COUNT(*)
FROM PEDIDO_PRODUCTO pp
LEFT JOIN PEDIDO p ON pp.id_pedido = p.id_pedido
```

```

WHERE p.id_pedido IS NULL

UNION ALL

SELECT 'PEDIDO_PRODUCTO → PRODUCTO', COUNT(*)
FROM PEDIDO_PRODUCTO pp
LEFT JOIN PRODUCTO pr ON pp.id_producto = pr.id_producto
WHERE pr.id_producto IS NULL

UNION ALL

SELECT 'ENVIO → PEDIDO', COUNT(*)
FROM ENVIO e
LEFT JOIN PEDIDO p ON e.id_pedido = p.id_pedido
WHERE p.id_pedido IS NULL;
  
```

VERIFICACIÓN 3: Cardinalidad - Pedidos sin Productos

Esta verificación valida que todo pedido tenga al menos un producto asociado. Un pedido sin productos carece de sentido en el contexto propuesto.

Para verificar esto, realizamos un LEFT JOIN entre la tabla PEDIDO y PEDIDO_PRODUCTO, y filtramos los casos donde no existe ningún producto asociado (WHERE pp.id_pedido IS NULL).

El resultado esperado es 0, si esta consulta devuelve algún registro, significa que se encontraron pedidos "vacíos" en la base de datos, lo cual indica un error en la lógica de generación de datos.

```

SELECT
  'Pedidos SIN productos' AS Verificación,
  COUNT(*) AS Cantidad_Incorrecta
FROM PEDIDO p
LEFT JOIN PEDIDO_PRODUCTO pp ON p.id_pedido = pp.id_pedido
WHERE pp.id_pedido IS NULL;
  
```

VERIFICACIÓN 4: Distribución de Productos por Pedido

Aquí analizamos la distribución de cuántos productos tiene cada pedido.

A fin de validar esto, primero, en la subconsulta, agrupamos por id_pedido y cuento cuántos productos tiene cada pedido, y luego en la consulta externa, agrupamos por esa cantidad y cuento cuántos pedidos tienen 1 producto, cuántos tienen 2, cuántos tienen 3, etc.

Además, calculamos el porcentaje que representa cada grupo respecto al total de pedidos. Cuando generamos los datos, buscamos que aproximadamente el 30% de los pedidos tuvieran 1 producto, el 40% tuvieran 2 productos, y el 30% restante tuvieran 3 productos. Esta verificación confirma se logró esa distribución.

```

SELECT
    productos_por_pedido AS 'Productos/Pedido',
    COUNT(*) AS Cantidad_Pedidos,
    CONCAT(ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM PEDIDO),
1), '%') AS Porcentaje
FROM (
    SELECT id_pedido, COUNT(*) AS productos_por_pedido
    FROM PEDIDO_PRODUCTO
    GROUP BY id_pedido
) dist
GROUP BY productos_por_pedido
ORDER BY productos_por_pedido;
  
```

VERIFICACIÓN 5: Cardinalidad - Envíos con Estado Inválido

Esta verificación valida que solo los pedidos que estén en estado "Despachado" o "Entregado" tengan un registro de envío.

Para esto, realizamos un JOIN entre ENVIO y PEDIDO, y luego filtramos los casos donde el estado del pedido no es "Despachado" ni "Entregado". Si encuentro algún registro, significa que hay envíos asociados a pedidos que no deberían tenerlos.

El resultado esperado es 0, ya que todos los envíos deben corresponder únicamente a pedidos que fueron efectivamente despachados o entregados.

```

SELECT
    'Envíos con estado inválido' AS Verificación,
    COUNT(*) AS Cantidad_Incorrecta
FROM ENVIO e
JOIN PEDIDO p ON e.id_pedido = p.id_pedido
WHERE p.estado_pedido NOT IN ('Despachado', 'Entregado');
  
```

VERIFICACIÓN 6: Rangos de Dominio - Precios Válidos

En esta verificación controlamos que los precios de los productos estén dentro de un rango razonable que consideramos a la hora de cargar los datos, en donde definimos que los precios válidos deben estar entre \$500 y \$50,000.

Por lo que buscamos validar todos aquellos los precios sean:

1. Menor a \$500 (demasiado baratos)
2. Mayor a \$50,000 (demasiado caros)

El resultado esperado para ambas consultas es 0.

```
SELECT
    'Productos con precio < 500' AS Verificación,
    COUNT(*) AS Cantidad
FROM PRODUCTO
WHERE precio_unitario < 500

UNION ALL

SELECT
    'Productos con precio > 50000',
    COUNT(*)
FROM PRODUCTO
WHERE precio_unitario > 50000;
```

VERIFICACIÓN 7: Integridad Matemática - Totales de Pedidos

Lo que buscamos validar con esta verificación, es que cada pedido tiene un campo total que almacena el monto total del pedido. Este total debe ser exactamente igual a la suma de todos los subtotales de los productos de ese pedido en la tabla PEDIDO_PRODUCTO.

La consulta calcula la suma de los subtotales para cada pedido y la compara con el total almacenado. Utilizamos ABS() para obtener el valor absoluto de la diferencia, considerando como aceptable una diferencia de hasta 0.01 para contemplar posibles errores de redondeo por el uso de decimales.

Si se encontraran pedidos donde la diferencia es mayor a 0.01, significa que hay una inconsistencia entre el total almacenado con la suma real de los productos.

El resultado esperado es 0 pedidos con totales incorrectos.

```

SELECT
  'Pedidos con total incorrecto' AS Verificación,
  COUNT(*) AS Cantidad_Incorrecta
FROM PEDIDO p
WHERE ABS(p.total - COALESCE((
  SELECT SUM(pp.subtotal)
  FROM PEDIDO_PRODUCTO pp
  WHERE pp.id_pedido = p.id_pedido
), 0)) > 0.01;
  
```

VERIFICACIÓN 8: Integridad Temporal - Fechas Coherentes de Envío

En esta última verificación se buscó validar la lógica de los envíos, donde la fecha_entrega siempre debe ser igual o posterior a la fecha_despacho.

La consulta busca registros en la tabla ENVIO donde fecha_entrega < fecha_despacho.

Si esta consulta devuelve algún registro, significa que hay datos inconsistentes en las fechas.

El resultado esperado es 0 envíos con fechas incoherentes.

```

SELECT
  'Envíos con fecha_entrega < fecha_despacho' AS Verificación,
  COUNT(*) AS Cantidad_Incorrecta
FROM ENVIO
WHERE fecha_entrega < fecha_despacho;
  
```

Etapa 3:

Diseño de Consultas con Utilidad Realista

Consulta 1: Top 5 productos más vendidos.

```
SELECT
    p.producto_nombre,
    SUM(pp.cantidad) AS total_vendido
FROM
    PRODUCTO p
JOIN
    PEDIDO_PRODUCTO pp ON p.id_producto = pp.id_producto
GROUP BY
    p.producto_nombre
ORDER BY
    total_vendido DESC
LIMIT 5;
```

Utilidad de negocio: Esta consulta es fundamental para la gestión de inventario y estrategias de marketing. Permite identificar los productos "estrella" para asegurar su stock, destacarlos en campañas publicitarias y analizar qué características los hacen tan populares para replicar su éxito en otros artículos.

Sentencia SQL: Usando las tablas `PEDIDO_PRODUCTO` que tiene la cantidad vendida de cada producto en cada pedido y `PRODUCTO`, que nos permite obtener el `producto_nombre`, agregamos todas las cantidades de la tabla `pedido_producto` y seguido de una cláusula GROUP BY para asegurar que la suma se haga por separado para cada `id_producto`. Por último le agregamos `p.producto_nombre` para que la base de datos sepa cómo agrupar las sumas. Luego, las ordenamos y limitamos al número deseado, en este caso 5.-

Consulta 2: Clientes Fieles (con +10 pedidos).

```

SELECT
    C.cliente_nombre,
    COUNT(p.id_pedido) AS numero_de_pedidos
FROM
    CLIENTE c
JOIN
    PEDIDO p ON c.id_cliente = p.id_cliente
GROUP BY
    c.cliente_nombre
HAVING
    COUNT(p.id_pedido) >= 10
ORDER BY
    numero_de_pedidos DESC;
  
```

Utilidad de negocio: Esta consulta identifica al segmento de clientes más leales. Con esta lista, el equipo de marketing puede lanzar campañas de fidelización, ofrecer descuentos exclusivos, accesos anticipados a productos o crear un programa de recompensas para agradecer y retener a su clientela más valiosa.

Sentencia SQL: Usando la tabla **PEDIDO** agrupando por **id_cliente**, unimos la tabla **PEDIDO** con **CLIENTE** para poder mostrar el nombre del cliente en lugar de solo su ID. Luego haciendo uso de **HAVING**:

- Agrupamos todos los pedidos por cliente y contamos cuántos hay en cada grupo usando **GROUP BY** y **COUNT**.
- Filtramos esos grupos para quedarnos solo con los que tengan un conteo de 10 o más usando **HAVING**.

Consulta 3: Top 5 Clientes con más dinero invertido en la tienda.

```
SELECT
    C.id_cliente,
    C.cliente_nombre,
    SUM(p.total) AS gasto_total
FROM
    CLIENTE c
JOIN
    PEDIDO p ON c.id_cliente = p.id_cliente
GROUP BY
    c.id_cliente, c.cliente_nombre
ORDER BY
    gasto_total DESC
LIMIT 5
```

Utilidad de negocio: Esta es una consulta de alto valor estratégico. Identifica a los clientes "VIP" por su gasto. Esta información es oro para personalizar ofertas y para fortalecer la relación con los clientes más rentables.

Sentencia SQL: Como necesitamos el gasto total por cliente, la mejor forma de obtenerlo es uniendo `CLIENTE` y `PEDIDO`, sumando la columna total. Luego `GROUP BY` descendiente y limitamos a 5.

Consulta 4: Provincias con Ventas > \$100k y su producto estrella.

```
-- CTE 1: Calcula el total vendido por provincia y filtra las que superan
los $100,000.
WITH VentasPorProvincia AS (
  SELECT
    l.provincia,
    SUM(p.total) AS total_ventas
  FROM
    LOCALIDADES l
  JOIN
    CLIENTE c ON l.id_localidad = c.id_localidad
  JOIN
    PEDIDO p ON c.id_cliente = p.id_cliente
  GROUP BY
    l.provincia
  HAVING
    SUM(p.total) > 100000
),

-- CTE 2: Ranking de los productos más vendidos dentro de cada provincia.
RankingProductosProvincia AS (
  SELECT
    l.provincia,
    pr.producto_nombre,
    ROW_NUMBER() OVER(PARTITION BY l.provincia ORDER BY
SUM(pp.cantidad) DESC) as ranking
  FROM
    LOCALIDADES l
  JOIN
    CLIENTE c ON l.id_localidad = c.id_localidad
  JOIN
    PEDIDO p ON c.id_cliente = p.id_cliente
  JOIN
    PEDIDO_PRODUCTO pp ON p.id_pedido = pp.id_pedido
  JOIN
    PRODUCTO pr ON pp.id_producto = pr.id_producto
  GROUP BY
    l.provincia, pr.producto_nombre
)
-- Consulta Final: Une las provincias con altas ventas con su producto #1.
SELECT
  vpp.provincia,
  vpp.total_ventas,
  rpp.producto_nombre AS producto_mas_vendido
FROM
  VentasPorProvincia vpp
JOIN
  RankingProductosProvincia rpp ON vpp.provincia = rpp.provincia
WHERE
  rpp.ranking = 1
ORDER BY
  vpp.total_ventas DESC;
```

Utilidad de negocio: Esta consulta es clave para la logística y la estrategia de marketing regional. Permite identificar los mercados provinciales más importantes (Ventas > \$100k) y entender la demanda específica de cada uno (su producto_mas_vendido). Con estos datos, se pueden optimizar las campañas publicitarias por región, ajustar el stock en los centros de distribución locales y tomar decisiones informadas sobre expansiones.

Sentencia SQL: Identifica las provincias con mayores ventas totales y dentro de cada una de ellas muestra cuál fue el producto más vendido. Para lograrlo se utilizan **CTEs** nuevamente, que son las tablas temporales definidas dentro de la misma consulta.

En la 1era **CTE**, llamada **VentasPorProvincia**, se calcula el total de ventas agrupado por provincia. Se realiza la unión entre las tablas **LOCALIDADES**, **CLIENTE** y **PEDIDO**, vinculando las claves foráneas correspondientes: **id_localidad** y **id_cliente**. Luego mediante la función **SUM(p, total)** se obtiene la suma del total vendido en cada provincia. Luego se agrupan los resultados y se filtran con **HAVING**, aquellas que tengan más de 100k. En la 2da **CTE**, llamada **RankingProductosProvincia**, se genera un ranking de los productos más vendidos en cada provincia, según la cantidad total comprada. Para eso se enlazan las tablas **LOCALIDADES**, **CLIENTE**, **PEDIDO**, **PEDIDO_PRODUCTO** y **PRODUCTO**, de modo que se pueda acceder tanto a la información geográfica como a los detalles de cada producto vendido. Se agrupan los resultados y se les asigna un rankin. Por último, la **consulta final**, combina las dos **CTE** anteriores. Se realiza un **JOIN** entre **VentasPorProvincia** y **RankingProductosProvincia** mediante el campo **provincia** para obtener, de cada provincia con ventas superiores a \$100k el producto que ocupa el 1er lugar del rankin. Se filtra el resultado y se ordena la salida de forma descendiente.

Creación de la Vista

Elección de vista:

Para la elección de la vista, buscamos una que combine **PEDIDO**, **ENVIO** y **PEDIDO_PRODUCTO**, lo cual es increíblemente útil. Esto permite a un operador o a un analista ver rápidamente toda la información relevante de un envío sin tener que escribir **JOIN** complejo cada vez. Básicamente, crea un “reporte” pre-construido.

La vista se llama **vista_Detalle_Envio**, y muestra para cada producto dentro de cada envío, la información del envío, del pedido y del cliente.

```

CREATE OR REPLACE VIEW Vista_Detalle_Envios AS
SELECT
    e.id_envio,
    e.numero_seguimiento,
    e.estado_envio,
    e.fecha_despacho,
    e.fecha_entrega,
    p.id_pedido,
    p.fecha_pedido,
    c.cliente_nombre,
    pr.producto_nombre,
    pp.cantidad,
    pp.subtotal
FROM
    ENVIO e
JOIN
    PEDIDO p ON e.id_pedido = p.id_pedido
JOIN
    CLIENTE c ON p.id_cliente = c.id_cliente
JOIN
    PEDIDO_PRODUCTO pp ON p.id_pedido = pp.id_pedido
JOIN
    PRODUCTO pr ON pp.id_producto = pr.id_producto
ORDER BY
    e.fecha_despacho DESC, e.id_envio, pr.producto_nombre;
  
```

Para ver la vista ejecutamos la siguiente sentencia:

```
SELECT * FROM Vista_Detalle_Envios WHERE id_envio = 1234;
```

Esta vista, llamada **Vista_Detalle_Envios**, reúne en una sola consulta toda la información relacionada con los envíos realizados.

Combina datos de varias tablas **ENVIO**, **PEDIDO**, **CLIENTE**, **PEDIDO_PRODUCTO** y **PRODUCTO** para mostrar, en un único resultado, los detalles completos de cada envío.

Incluye información del envío (ID, número de seguimiento, estado, fechas de despacho y entrega), del pedido asociado (ID y fecha), del cliente (nombre), y de los productos enviados (nombre, cantidad y subtotal).

La cláusula **ORDER BY e.fecha_despacho DESC** ordena los resultados mostrando primero los envíos más recientes.

En resumen, esta vista facilita consultar el historial completo de envíos con todos los datos relevantes del cliente, pedido y productos en una sola estructura.

Creación de Índices y Mediciones de Tiempos

Primera Consulta:

```
SELECT * FROM PEDIDO WHERE id_cliente = 15000;
```

Tiempos SIN Índice:

Mediciones:

Todas las mediciones fueron para estos resultados:

	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	id_pedido	id_cliente	fecha_pedido	estado_pedido	total	observaciones
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	11030	15000	2025-04-07 00:00:00	Pendiente	73972.51	Envío urgente
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	39550	15000	2024-12-30 00:00:00	Despachado	142501.63	Envío urgente
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	47308	15000	2025-09-20 00:00:00	Cancelado	85987.32	Envío urgente
<input type="checkbox"/>	<input type="button" value="Edit"/>	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>	83916	15000	2025-06-05 00:00:00	En Preparacion	34092.68	NULL

1.

 Showing rows 0 - 3 (4 total, Query took 0.0701 seconds.)

```
SELECT * FROM PEDIDO WHERE id_cliente = 15000;
```

2.

 Showing rows 0 - 3 (4 total, Query took 0.0505 seconds.)

```
SELECT * FROM PEDIDO WHERE id_cliente = 15000;
```

3.

 Showing rows 0 - 3 (4 total, Query took 0.0483 seconds.)

```
SELECT * FROM PEDIDO WHERE id_cliente = 15000;
```

Tiempo mediano SIN ÍNDICE: 0.0505 segundos.

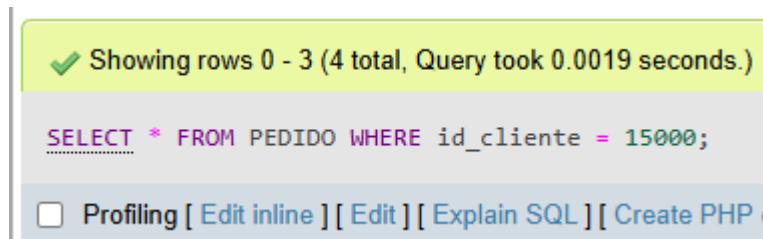
Tiempos CON Índice

Creamos el índice:

```
CREATE INDEX idx_pedido_cliente ON PEDIDO(id_cliente);
```

Mediciones:

1.

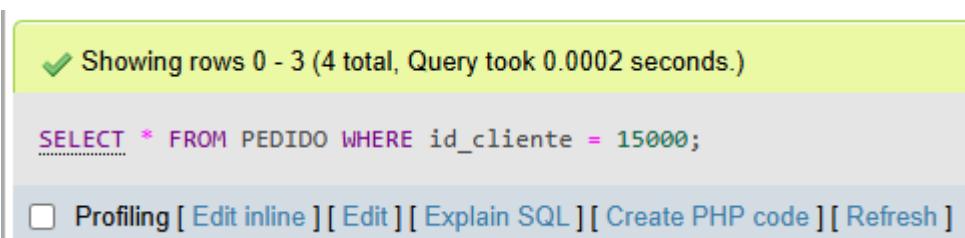


Showing rows 0 - 3 (4 total, Query took 0.0019 seconds.)

```
SELECT * FROM PEDIDO WHERE id_cliente = 15000;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

2.

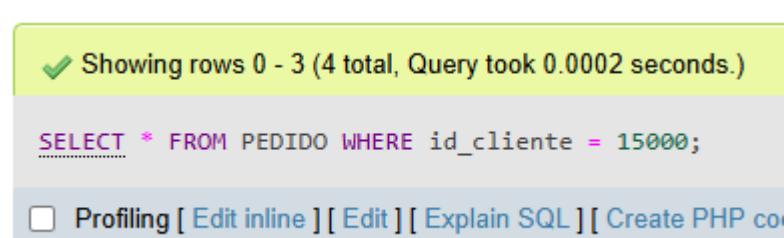


Showing rows 0 - 3 (4 total, Query took 0.0002 seconds.)

```
SELECT * FROM PEDIDO WHERE id_cliente = 15000;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

3.



Showing rows 0 - 3 (4 total, Query took 0.0002 seconds.)

```
SELECT * FROM PEDIDO WHERE id_cliente = 15000;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Tiempo mediano CON ÍNDICE: 0.0002 segundos.

Segunda Consulta:

```
SELECT * FROM PEDIDO WHERE fecha_pedido BETWEEN '2025-01-01' AND  
'2025-03-31';
```

Tiempos SIN Índice:

Mediciones:

1. .

✓ Showing rows 0 - 24 (24774 total, Query took 0.0007 seconds.)

```
SELECT * FROM PEDIDO WHERE fecha_pedido BETWEEN '2025-01-01' AND '2025-03-31';
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

2. .

✓ Showing rows 0 - 24 (24774 total, Query took 0.0005 seconds.)

```
SELECT * FROM PEDIDO WHERE fecha_pedido BETWEEN '2025-01-01' AND '2025-03-31';
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

3. .

✓ Showing rows 0 - 24 (24774 total, Query took 0.0003 seconds.)

```
SELECT * FROM PEDIDO WHERE fecha_pedido BETWEEN '2025-01-01' AND '2025-03-31';
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Tiempo mediano SIN ÍNDICE: 0.0005 segundos.

Tiempos CON Índice

Creamos el índice:

```
CREATE INDEX idx_pedido_fecha ON PEDIDO(fecha_pedido);
```

Mediciones:

1. .

```
Showing rows 0 - 24 (24774 total, Query took 0.0064 seconds.)  
  
SELECT * FROM PEDIDO WHERE fecha_pedido BETWEEN '2025-01-01' AND '2025-03-31';  
  
 Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]
```

2. .

```
Showing rows 0 - 24 (24774 total, Query took 0.0003 seconds.)  
  
SELECT * FROM PEDIDO WHERE fecha_pedido BETWEEN '2025-01-01' AND '2025-03-31';  
  
 Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]
```

3. .

```
Showing rows 0 - 24 (24774 total, Query took 0.0003 seconds.)  
  
SELECT * FROM PEDIDO WHERE fecha_pedido BETWEEN '2025-01-01' AND '2025-03-31';  
  
 Profiling [ Edit inline ] [ Edit ] [ Explain SQL ] [ Create PHP code ] [ Refresh ]
```

Tiempo mediano CON ÍNDICE: 0.0003 segundos.

De primeras cuesta hacerlo correr porque carga los datos en la memoria del equipo.
Por eso aunque no tengan índice, las segundas consultas son mucho más rápidas.

Tercera Consulta: Join con Group By.-

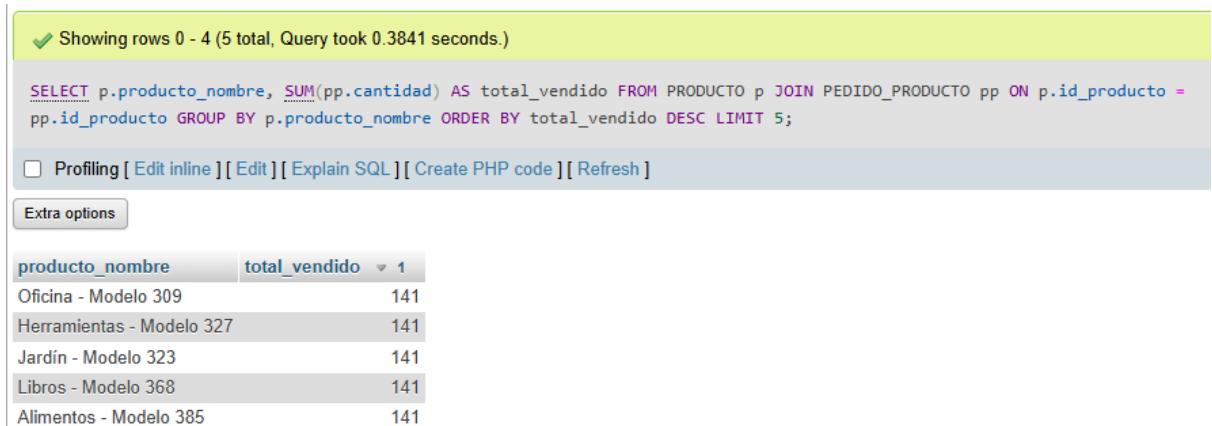
```

SELECT
    p.producto_nombre,
    SUM(pp.cantidad) AS total_vendido
FROM
    PRODUCTO p
JOIN
    PEDIDO_PRODUCTO pp ON p.id_producto = pp.id_producto
GROUP BY
    p.producto_nombre
ORDER BY
    total_vendido DESC
LIMIT 5;
  
```

Tiempos SIN Índice:

Mediciones:

1. .



Showing rows 0 - 4 (5 total, Query took 0.3841 seconds.)

```

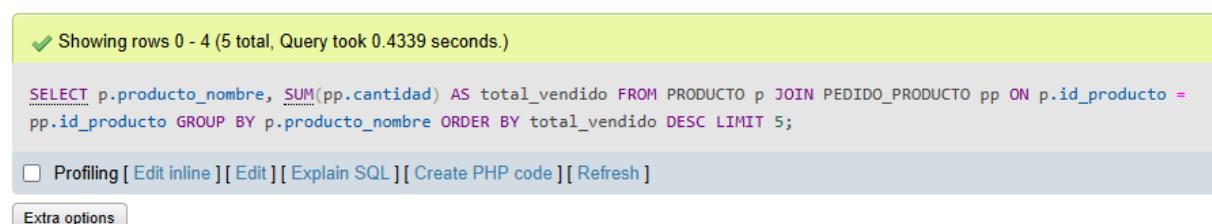
SELECT p.producto_nombre, SUM(pp.cantidad) AS total_vendido FROM PRODUCTO p JOIN PEDIDO_PRODUCTO pp ON p.id_producto = pp.id_producto GROUP BY p.producto_nombre ORDER BY total_vendido DESC LIMIT 5;
  
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Extra options

producto_nombre	total_vendido
Oficina - Modelo 309	141
Herramientas - Modelo 327	141
Jardín - Modelo 323	141
Libros - Modelo 368	141
Alimentos - Modelo 385	141

2. .



Showing rows 0 - 4 (5 total, Query took 0.4339 seconds.)

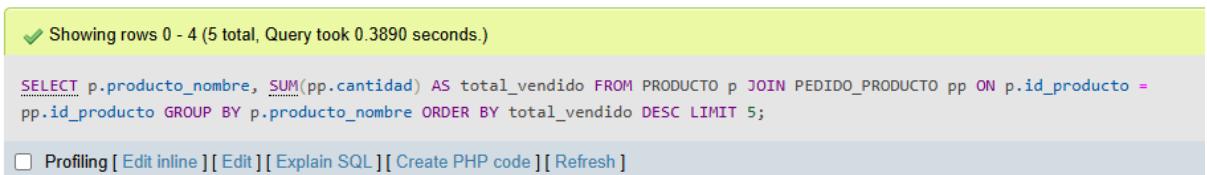
```

SELECT p.producto_nombre, SUM(pp.cantidad) AS total_vendido FROM PRODUCTO p JOIN PEDIDO_PRODUCTO pp ON p.id_producto = pp.id_producto GROUP BY p.producto_nombre ORDER BY total_vendido DESC LIMIT 5;
  
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Extra options

3. .



Showing rows 0 - 4 (5 total, Query took 0.3890 seconds.)

```

SELECT p.producto_nombre, SUM(pp.cantidad) AS total_vendido FROM PRODUCTO p JOIN PEDIDO_PRODUCTO pp ON p.id_producto = pp.id_producto GROUP BY p.producto_nombre ORDER BY total_vendido DESC LIMIT 5;
  
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Tiempo mediano SIN ÍNDICE: 0.3890 segundos.

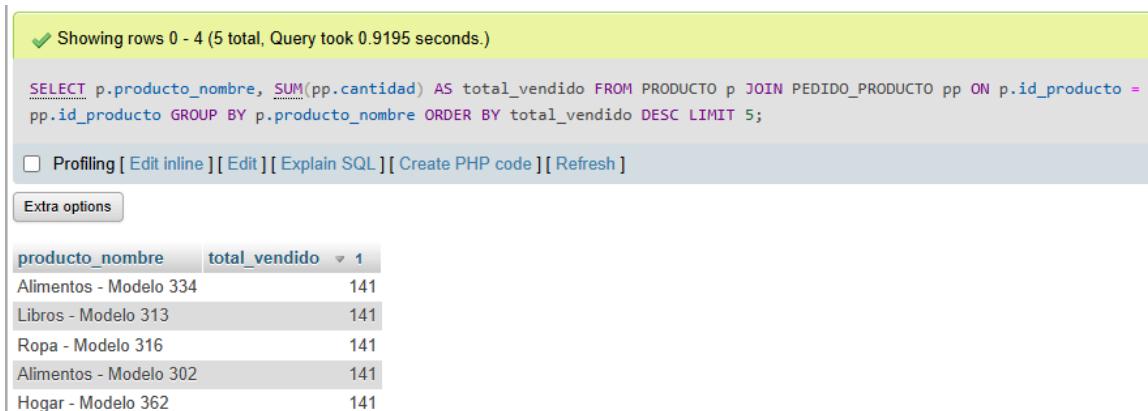
Tiempos CON Índice

Creamos el índice:

```
CREATE INDEX idx_pp_producto ON PEDIDO_PRODUCTO(id_producto);
```

Mediciones:

1. .



Showing rows 0 - 4 (5 total, Query took 0.9195 seconds.)

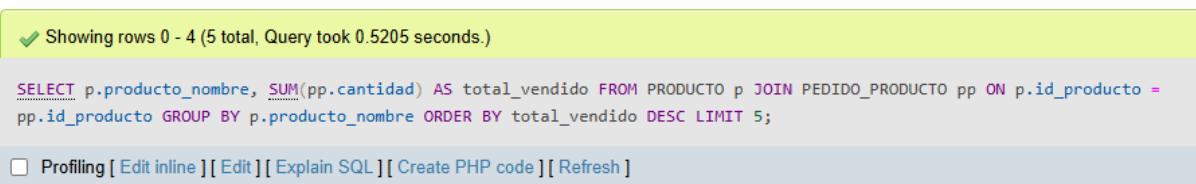
```
SELECT p.producto_nombre, SUM(pp.cantidad) AS total_vendido FROM PRODUCTO p JOIN PEDIDO_PRODUCTO pp ON p.id_producto = pp.id_producto GROUP BY p.producto_nombre ORDER BY total_vendido DESC LIMIT 5;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Extra options

producto_nombre	total_vendido
Alimentos - Modelo 334	141
Libros - Modelo 313	141
Ropa - Modelo 316	141
Alimentos - Modelo 302	141
Hogar - Modelo 362	141

2.

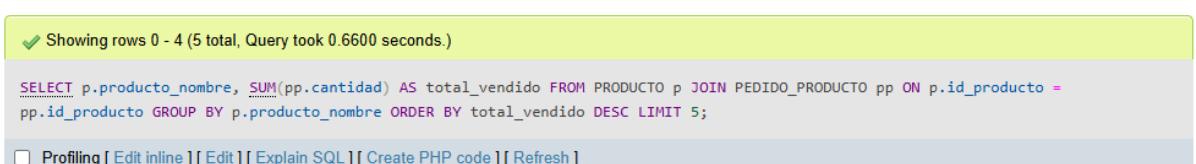


Showing rows 0 - 4 (5 total, Query took 0.5205 seconds.)

```
SELECT p.producto_nombre, SUM(pp.cantidad) AS total_vendido FROM PRODUCTO p JOIN PEDIDO_PRODUCTO pp ON p.id_producto = pp.id_producto GROUP BY p.producto_nombre ORDER BY total_vendido DESC LIMIT 5;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

3. .



Showing rows 0 - 4 (5 total, Query took 0.6600 seconds.)

```
SELECT p.producto_nombre, SUM(pp.cantidad) AS total_vendido FROM PRODUCTO p JOIN PEDIDO_PRODUCTO pp ON p.id_producto = pp.id_producto GROUP BY p.producto_nombre ORDER BY total_vendido DESC LIMIT 5;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Tiempo mediano CON ÍNDICE: 0.6600 segundos.

Conclusión del estudio:

Más allá del uso de índices, esta etapa demostró que la optimización del rendimiento es una tarea de análisis contextual. Si bien un índice aceleró drásticamente una búsqueda por igualdad (más de 250 veces), su efectividad no fue universal. En consultas de rango amplio y en agregaciones que recorrían tablas completas, el índice no ofreció mejoras e incluso, en el caso del **JOIN**, resultó ser más lento.

Aprendimos que el optimizador de MySQL es lo suficientemente inteligente para preferir un escaneo completo de la tabla cuando la consulta no es **selectiva**, es decir, cuando debe procesar un gran porcentaje de los datos. La lección principal es que un índice no es una solución mágica, sino una herramienta cuyo valor depende críticamente de la naturaleza de la consulta a optimizar, demostrando que a veces la estrategia más simple es la más eficiente.

Explain de las consultas:

Repetimos las dos últimas consultas ahora con **EXPLAIN** para ver sus resultados.

Consulta 2:

Con Índice:

Your SQL query has been executed successfully.

```
EXPLAIN SELECT * FROM PEDIDO WHERE fecha_pedido BETWEEN '2025-01-01' AND '2025-03-31';
```

[Edit inline] [Edit] [Skip Explain SQL] [Create PHP code]

Extra options

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	PEDIDO	ALL	idx_pedido_fecha	NULL	NULL	NULL	99771	Using where

Sin Índice:

Your SQL query has been executed successfully.

```
EXPLAIN SELECT * FROM PEDIDO WHERE fecha_pedido BETWEEN '2025-01-01' AND '2025-03-31';
```

[Edit inline] [Edit] [Skip Explain SQL] [Create PHP code]

Extra options

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	PEDIDO	ALL	NULL	NULL	NULL	NULL	99771	Using where

Consulta 3:

Con Índice:

Your SQL query has been executed successfully.

```
EXPLAIN SELECT p.producto_nombre, SUM(pp.cantidad) AS total_vendido FROM PRODUCTO p JOIN PEDIDO_PRODUCTO pp ON p.id_producto = pp.id_producto GROUP BY p.producto_nombre ORDER BY total_vendido DESC LIMIT 5;
```

[Edit inline] [Edit] [Skip Explain SQL] [Create PHP code]

[Extra options](#)

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	p	ALL	PRIMARY	NULL	NULL	NULL	4985	Using temporary; Using filesort
1	SIMPLE	pp	ref	idx_pp_producto	idx_pp_producto	4	tpi_pedido_envio_test.p.id_producto	19	

Sin Índice:

Your SQL query has been executed successfully.

```
EXPLAIN SELECT p.producto_nombre, SUM(pp.cantidad) AS total_vendido FROM PRODUCTO p JOIN PEDIDO_PRODUCTO pp ON p.id_producto = pp.id_producto GROUP BY p.producto_nombre ORDER BY total_vendido DESC LIMIT 5;
```

[Edit inline] [Edit] [Skip Explain SQL] [Create PHP code]

[Extra options](#)

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	pp	ALL	NULL	NULL	NULL	NULL	199826	Using temporary; Using filesort
1	SIMPLE	p	eq_ref	PRIMARY	PRIMARY	4	tpi_pedido_envio_test.pp.id_producto	1	

Etapa 4:

Creación de usuario con permisos acotados (principio de mínimos privilegios)

Definimos dos usuarios a modo de evidenciar la regla de usuario con mínimos privilegios y otro con privilegios de root.

En este trabajo nos reservamos el uso de los usuarios root por defecto como una buena práctica, para evitar errores a la hora de otorgar y quitar privilegios.

Los usuarios creados son:

- **user_ventas**: empleados en el área de venta, con privilegios mínimos para ver información pública y realizar pedidos.
- **admin_bd**: usuario que se encargaría del manejo y administración de la base de datos, para su mantenimiento y en caso de eliminaciones o errores, poder acudir a ella. Sus privilegios son de root.

El usuario '**user_ventas**' pertenece a un empleado activo con una contraseña propia, sus privilegios mínimos son ver vistas públicas y registrar pedidos, no debería acceder a información sensible.

Una medida de seguridad es otorgar permisos sobre vista y no sobre las tablas bases, ej: **vista_cliente_publico**.

El **INSERT** en **PEDIDO** y **PEDIDO_PRODUCTO** permite registrar ventas sin exponer datos sensibles ni dar poder de borrado/alteración.

```
-- Crear usuario con minimos privilegios
CREATE USER 'user_ventas'@'localhost' IDENTIFIED BY '1234';

-- Quitar cualquier privilegio por defecto (pro medida de seguridad)
-- REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'user_ventas'@'localhost';

-- Otorgar privilegios necesarios:
-- SELECT sobre las vistas (que crearemos abajo)
-- INSERT sobre PEDIDO y PEDIDO_PRODUCTO para poder crear pedidos
GRANT SELECT ON tpi_pedido_envio.vista_clientes_publicos TO 'user_ventas'@'localhost';
GRANT SELECT ON tpi_pedido_envio.vista_pedidos_resumen TO 'user_ventas'@'localhost';

GRANT INSERT ON tpi_pedido_envio.PEDIDO TO 'user_ventas'@'localhost';
GRANT INSERT ON tpi_pedido_envio.PEDIDO_PRODUCTO TO 'user_ventas'@'localhost';

-- No damos UPDATE/DELETE/ALTER/DROP ni acceso directo a tablas sensibles.
FLUSH PRIVILEGES;
```

El usuario '**admin_db**', su función es la manipulación, administración de los datos y el poder de otorgar a otros usuario privilegios.

```

-- Una forma menos recomendadas es que tenga los permisos sobre las tablas y columnas de la DB.
CREATE USER 'admin_db'@'localhost' IDENTIFIED BY '0000'; -- Creacion del usuario admin_db (USAGE = sin privilegios
GRANT ALL privileges ON *.* TO 'admin_db'@'localhost' WITH GRANT OPTION;
-- WITH GRANT OPTION permite que este usuario otorgue privilegios a otros
-- Tener en cuenta que los privilegios solo son aplicables a tpi_pedido_envio
FLUSH PRIVILEGES; -- aplica los cambios inmediatamente

```

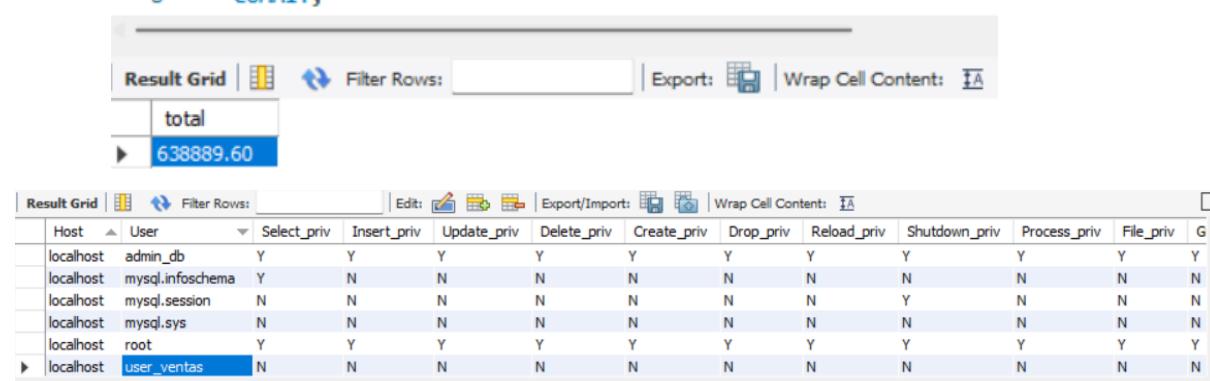
En la siguiente imagen se evidencia la vista de una tabla compuesta por todos los usuarios existentes, es de interés observar **user_ventas** el cual muestra en su fila que no tiene privilegios sobre las tablas.

En cambio el otro usuario creado, '**admin_bd**' cuenta con script que le da privilegios de root, lo recomendable no es ocupar este último para evitar errores a la hora de otorgar privilegios.

```

1      -- Nivel READ COMMITED
2 •  SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
3
4 •  START transaction;
5 •  SELECT total FROM PEDIDO WHERE id_pedido = 1;
6      -- otra sesion actualizada total:
7      -- UPDATE PEDIDO SET totla = total + 50 WHERE id_pedido = 1
8 •  COMMIT;

```



Host	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Reload_priv	Shutdown_priv	Process_priv	File_priv	G
localhost	admin_db	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
localhost	mysql.infoschema	Y	N	N	N	N	N	N	N	N	N	N
localhost	mysql.session	N	N	N	N	N	N	Y	N	N	N	N
localhost	mysql.sys	N	N	N	N	N	N	N	N	N	N	N
localhost	root	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
localhost	user_ventas	N	N	N	N	N	N	N	N	N	N	N

Puntos a mejorar en un caso real:

Creación de más usuarios con distintos privilegios de manera escalonada (empleado, supervisor, gerente, admin_bd_envio, etc.), de esta forma no tendría control total una persona evitando errores masivos y también distribuyendo la carga de trabajo en el área de mantenimiento de las bases.

Referencias: anexo, Ref 1d

Vistas que ocultan información sensible

Con el siguiente script, pudimos crear una vista sin los email ni las direcciones de los clientes, lo cual es considerado información sensible.

```
DROP VIEW IF EXISTS tpi_pedido_envio.vista_clientes_publicos;
CREATE SQL SECURITY DEFINER VIEW tpi_pedido_envio.vista_clientes_publicos AS
SELECT
  id_cliente,
  cliente_nombre,
  cliente_telefono,
  id_localidad
FROM tpi_pedido_envio.CLIENTE;
```

Obteniendo una vista sesgada de la tabla **tpi_pedido_envio**. Lo importante es que protegemos datos sensibles de los clientes.

	id_cliente	cliente_nombre	cliente_telefono	id_localidad
▶	1	Juan Pérez	11-4567-8900	1
	2	Juan Moreno 90	11-2302-5767	21
	3	Juan Moreno 91	11-8286-7892	25
	4	Juan Moreno 92	11-5934-3067	33
	5	Juan Moreno 93	11-5474-5034	31
	6	Juan Moreno 94	11-9046-5925	1
	7	Juan Moreno 95	11-1771-4761	25
	8	Juan Moreno 96	11-4947-6773	20
	9	Juan Moreno 97	11-1322-9562	45
	10	Juan Moreno 98	11-6313-2409	38
	11	Juan Moreno 99	11-2274-4033	11

vista_clientes_publicos 4 ×

En este caso el script que ejecuta crea una vista, llamada “**vista_pedidos_resumen**”, la cual está sesgada sin las direcciones de entrega ni datos de seguimiento.

```
DROP VIEW IF EXISTS tpi_pedido_envio.vista_pedidos_resumen;
CREATE SQL SECURITY DEFINER VIEW tpi_pedido_envio.vista_pedidos_resumen AS
SELECT
  p.id_pedido,
  p.id_cliente,
  c.cliente_nombre,
  p.fecha_pedido,
  p.estado_pedido,
  p.total,
  p.observaciones
FROM tpi_pedido_envio.PEDIDO p
JOIN tpi_pedido_envio.CLIENTE c ON p.id_cliente = c.id_cliente;
```

A continuación vemos la **vista_pedido_resumen**, realizada con la función **SELECT**.

	Result Grid	Filter Rows:	Export:	Wrap Cell Content:	Fetch rows:		
	id_pedido	id_cliente	cliente_nombre	fecha_pedido	estado_pedido	total	observaciones
▶	1	1	Juan Pérez	2024-10-15 10:30:00	Confirmado	37000.00	Entrega en horario comercial
	2	1	Juan Pérez	2024-10-16 09:15:30	Despachado	50000.00	Entregar en portería
	3	7422	Carlos Rodríguez 70	2025-01-12 00:00:00	Pendiente	0.00	Envío urgente
	4	2317	Juan Rodríguez 185	2025-04-14 00:00:00	En Preparacion	0.00	NULL
	5	13572	Laura Díaz 120	2025-06-29 00:00:00	Confirmado	0.00	Envío urgente
	6	27277	Lucas Pérez 25	2025-08-01 00:00:00	En Preparacion	0.00	Envío urgente
	7	6689	Carlos García 17	2025-07-12 00:00:00	Entregado	0.00	NULL
	8	24627	Valeria Pérez 75	2025-02-07 00:00:00	Pendiente	0.00	Envío urgente
	9	15140	Laura Fernández 168	2025-09-02 00:00:00	Pendiente	0.00	NULL
	10	18090	Diego González 18	2025-01-19 00:00:00	Cancelado	0.00	NULL
	11	79445	Lucas González 53	2025-10-10 00:00:00	Confirmado	0.00	NULL

A continuación con el siguiente script subrayado en azul, hacemos uso de la tabla **tpi_pedido_envio** y mostramos todas las tablas donde el tipo de tabla es 'VIEW' (vista).

Y de esta forma evidenciamos que se crearon correctamente las vistas antes mencionadas.

```

1 • USE tpi_pedido_envio;
2 • SHOW FULL TABLES IN tpi_pedido_envio WHERE TABLE_TYPE = 'VIEW';
3 • show tables;
4 -- Vista de clientes pública (sin email ni dirección)
5 • DROP VIEW IF EXISTS tpi_pedido_envio.vista_clientes_publicos;
```

	Result Grid	Filter Rows:	Export:	Wrap Cell Content:
	Tables_in_tpi_pedido_envio	Table_type		
▶	vista_clientes_publicos	VIEW		
	vista_pedidos_resumen	VIEW		

Puntos a tener en cuenta o de mejora

Concepto importante:

SQL SECURITY DEFINER es una cláusula que especifica que un objeto de base de datos (como una función, procedimiento o vista) se ejecutará con los privilegios del usuario que lo creó y no del usuario que lo llama.

Otra forma es la ejecución con privilegios del llamante, **SECURITY INVOKER**, es decir que se ejecuta con los permisos del usuario que las llama, siendo esta utilizada por defecto.

Referencias: Anexo Ref 2

Violacion de restricciones (PK, FK, UNIQUE, CHECK)

Violacion de PK (duplicado de llave primaria)

En este caso para evidenciar el buen funcionamiento de las restricciones aplicadas a la llave primaria **id_producto**, debemos ingresar un dato que ya esté en la base de esta forma va contra una de los principios de las llaves primarias, que sean únicas.

La inserción que vamos a realizar es la siguiente:

- Suponiendo que **id_producto** es 1.
- Intentó forzar duplicado en PRODUCTO (falla si **id_producto** es PK y AUTO_INCREMENT)

```
INSERT INTO localidades(id_localidad,ciudad,provincia,codigo_postal)
VALUES (1,'Ciudad Invetada', 'Provincia tambien', '123');
```

El error es el esperado: **ERROR 1062 (23000): Duplicate entry '1' for key 'localidades.PRIMARY'**.

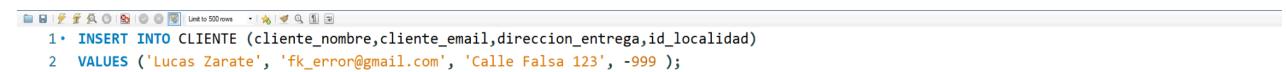
MySQL said: 

```
#1062 - Duplicate entry '1' for key 'localidades.PRIMARY'
```

Violacion de FK (referencia a cliente inexistente)

Se intenta la inserción de un **pedido** con **id_cliente** inexistente, por ejemplo: **999999**.

```
INSERT INTO CLIENTE
(cliente_nombre,cliente_email,direccion_entrega,id_localidad)
VALUES
('Lucas Zarate', 'fk_error@gmail.com', 'Calle Falsa 123', -999 );
```



1. INSERT INTO CLIENTE (cliente_nombre,cliente_email,direccion_entrega,id_localidad)
2. VALUES ('Lucas Zarate', 'fk_error@gmail.com', 'Calle Falsa 123', -999);



Output

#	Time	Action
1	19:33:21	INSERT INTO CLIENTE (cliente_nombre,cliente_email,direccion_entrega,id_localidad) VALUES ('Lucas Zarate', 'fk_error@gmail.com', 'Calle Falsa 123', -999)

Message

Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (*'tbl_pedido_envio'.cliente*.CONSTRAINT fk_cliente_localidad FOREIGN KEY (*id_localidad*) REFERENCES localidades (*id_localidad*))

Error esperado: ERROR 1452: Cannot add or update a child row: a foreign key constraint fails

La razón es porque **fk_pedido_cliente** que obliga a que **id_cliente** exista en **CLIENTE**.

Violacion de UNIQUE (código de producto duplicado)

Para hacer ver como trabaja la restricción **UNIQUE**, respetando que no haya duplicados en este caso en **producto_codigo**, ejecutamos dos inserciones de datos para un mismo código “**codigo_001**”.

```
INSERT INTO PRODUCTO (producto_nombre, producto_codigo, precio_unitario, stock_disponible)
VALUES ('Notebook_duplicada', 'codigo_001', 100.00, 10);

-- intentar duplicar codigo
INSERT INTO PRODUCTO (producto_nombre, producto_codigo, precio_unitario, stock_disponible)
VALUES ('Notebook_duplicada', 'codigo_001', 50.00, 5);
```

El error a continuación es el esperado, en este se evidencia el duplicado.

MySQL said: 

```
#1062 - Duplicate entry 'codigo_001' for key 'producto.uk_producto_codigo'
```

La razón de error es por que constraint **UNIQUE** evita código repetidos.

Violacion de CHECK (precio negativo, cantidad inválida)

En este caso vamos a enfrentarnos a la restricción **CHECK**, su función es el chequeo de parámetros impuestos por nosotros.

Por ejemplo, vamos a hacer el intento de cargar un precio negativo a un producto.

Adjunto donde se halla el **CHECK** que vamos a probar:

```
CREATE TABLE PEDIDO_PRODUCTO (
    id_detalle INT AUTO_INCREMENT PRIMARY KEY,
    id_pedido INT NOT NULL,
    id_producto INT NOT NULL,
    cantidad INT NOT NULL,
    precio_unitario DECIMAL(10,2) NOT NULL,
    subtotal DECIMAL(10,2) NOT NULL,
    CONSTRAINT fk_detalle_pedido FOREIGN KEY (id_pedido)
        REFERENCES PEDIDO(id_pedido)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT fk_detalle_producto FOREIGN KEY (id_producto)
        REFERENCES PRODUCTO(id_producto)
        ON DELETE RESTRICT
        ON UPDATE CASCADE,
    CONSTRAINT chk_detalle_cantidad CHECK (cantidad > 0),
    CONSTRAINT chk_detalle_precio CHECK (precio_unitario >= 0),
    CONSTRAINT chk_detalle_subtotal CHECK (subtotal >= 0),
    CONSTRAINT chk_detalle_calculo CHECK (ABS(subtotal - (cantidad * precio_unitario)) < 0.01)
```

A continuación el script con **precio_unitario = -102.00** que ejecutamos es el siguiente:

```
INSERT INTO PRODUCTO (producto_nombre, producto_codigo, precio_unitario, stock_disponible)
VALUES ('Tele_regalado', '001', -102.00, 1);
```

El error es siguiente:

MySQL said: 

```
#3819 - Check constraint 'chk_producto_precio' is violated.
```

El error indica que el **CHECK CONSTRAINT** su violado es decir que no cumple con la condiciones impuestas (**precio_unitario >=0**).

Implementar una consulta segura (JAVA con PreparedStatement) + prueba anti-SQL injection

PreparedStatement separa la consulta de los datos. El '?' funciona como un elemento temporal que espera a que se introduzca un contenido definitivo (placeholder), cuando hace `ps.setString(1, intento_malisioso)`, el driver escapa y trata toda la cadena como dato sin permitir que contenga fragmentos que cambien la estructura SQL.

Nuestro punto de partida, lo conocemos con la función `SELECT COUNT(*) FROM PRODUCTO;`; en este caso vamos a experimentar con la tabla producto.

Lo que nos devuelve el script es la cantidad de 5006 datos:

Result Grid	
	COUNT(*)
▶	5006

Si el script realizado en java funciona correctamente, este número no debería cambiar ya que vamos a estar tratando de borrar una tabla completa.

```
USE tpi_pedido_envio;
INSERT INTO producto (producto_nombre, producto_codigo, precio_unitario, stock_disponible)
values ('','ps3',101); DROP TABLE PRODUCTO; --,4)
```

A continuación ejecutamos el siguiente script que busca hacer una inserción de un producto, con intenciones maliciosas.

En la valor de **precio_unitario**, (101; DROP TABLE PRODUCTO; --, 4) se busca eliminar de forma permanente la tabla PRODUCTO, con sus datos, disparadores, y restricciones asociadas.

```
Output - tpi_bases_de_datos1 (run) ×
run:
Insercion legitima OR
java.sql.SQLIntegrityConstraintViolationException: Duplicate entry '1; DROP TABLE PRODUCTO; --' for key 'producto.uk_producto_codigo'
  at com.mysql.cj.jdbc.exceptions.SQLException.createSQLException(SQLException.java:109)
  at com.mysql.cj.jdbc.exceptions.SQLExceptionsMapping.translateException(SQLExceptionsMapping.java:114)
  at com.mysql.cj.jdbc.ClientPreparedStatement.executeInternal(ClientPreparedStatement.java:988)
  at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdateInternal(ClientPreparedStatement.java:1166)
  at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdateInternal(ClientPreparedStatement.java:1101)
  at com.mysql.cj.jdbc.ClientPreparedStatement.executeLargeUpdate(ClientPreparedStatement.java:1448)
  at com.mysql.cj.jdbc.ClientPreparedStatement.executeUpdate(ClientPreparedStatement.java:1084)
  at tpi_bases_de_datos1.dbConnection_anti_inyeccion.main(dbConnection anti inyeccion.java:36)
BUILD SUCCESSFUL (total time: 3 seconds)
```

Este error nos indica que el intento de inyección falló, el PreparedStatement trato la cadena ('1;DROP TABLE PRODUCTO; --') como un valor literal, no como una cadena ejecutable.

Demostrando así la inyección neutralizada.

Quedando en consideración que el error no lo causa el DROP TABLE propiamente, sino la restricción UNIQUE sobre **producto_codigo(uk_producto_codigo)**. Y como fue manipulado como una cadena de texto, se encontró que ya existía un código igual o como duplicado en la columna con UNIQUE. Por eso el error es Duplicate entry

ERROR: java.sql.SQLIntegrityConstraintViolationException: Duplicate entry '1; DROP TABLE PRODUCTO; --' for key 'producto.uk_producto_codigo'

Por último a modo de chequear que la tabla PRODUCTO realmente no fue eliminada, realizamos la siguiente consulta:

5 SELECT * FROM PRODUCTO;

Result Grid | Filter Rows: | Edit: | Export/Import: |

	id_producto	producto_nombre	producto_codigo	precio_unitario	stock_disponible
▶	1	Notebook Lenovo IdeaPad	NB-LEN-001	450000.00	11
	2	Mouse Logitech M185	MS-LOG-185	8500.00	47
	3	Mouse Logitech G203	MS-LOG-203	3500.00	30
	4	Electrónica - Modelo 410	PROD-000410	3590.59	50
	5	Electrónica - Modelo 310	PROD-000310	4308.32	125
	6	Electrónica - Modelo 210	PROD-000210	45042.82	124
	7	Electrónica - Modelo 110	PROD-000110	20538.79	32
	8	Electrónica - Modelo 10	PROD-000010	29851.55	96
	9	Laptop Dell XPS 13	DELL-XPS-13	21220.22	124
	10	Laptop HP Pavilion 15	HP-15-2020	25000.00	124

Etapa 5

Simulación de deadlock

Deadlock o punto muerto: es una instancia donde 2 o más transacciones se encuentran en espera, con la particularidad de que ambas se encuentran bloqueadas (mecanismo de control utilizado en la concurrencia)

Las soluciones que existen a este problema son:

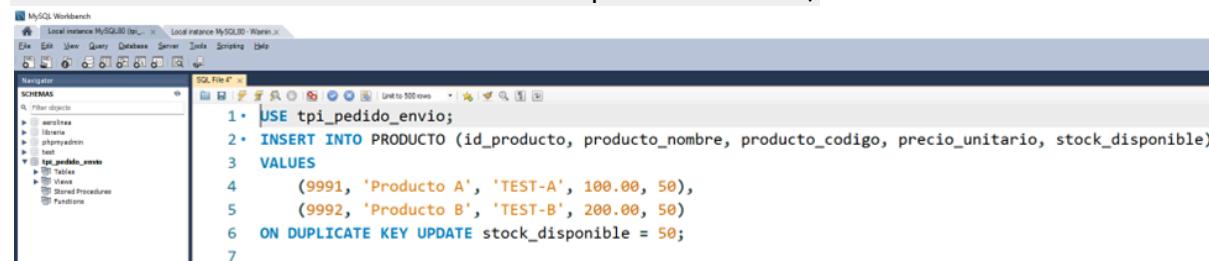
- Algoritmos de detección de bloqueos.
- Selección de víctimas (que transacción abortar)
- Retroceso y reintento automáticos
- Mecanismos de tiempo de espera.

Simulación de deadlock:

Preparamos **PRODUCTO** para la prueba:

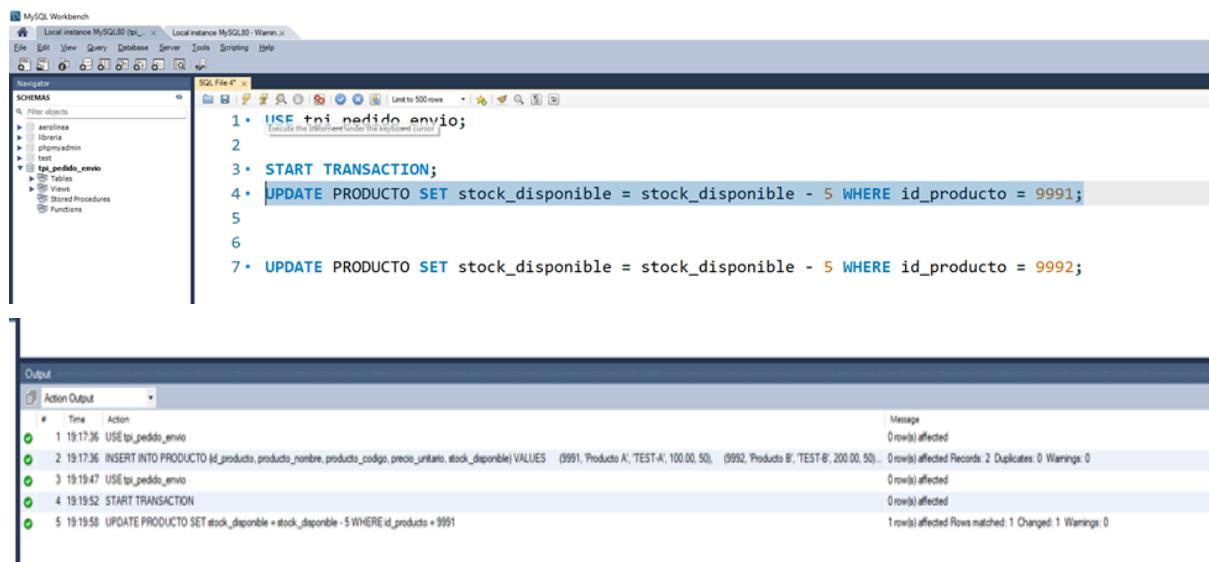
SESION 1

```
USE tpi_pedido_envio;
INSERT INTO PRODUCTO (id_producto, producto_nombre, producto_codigo,
precio_unitario, stock_disponible)
VALUES
    (9991, 'Producto A', 'TEST-A', 100.00, 50),
    (9992, 'Producto B', 'TEST-B', 200.00, 50)
ON DUPLICATE KEY UPDATE stock_disponible = 50;
```



SESION 1

```
USE tpi_pedido_envio;
START TRANSACTION;
UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5
WHERE id_producto = 9991;
```



The screenshot shows the MySQL Workbench interface with a script editor and an output window.

Script Editor:

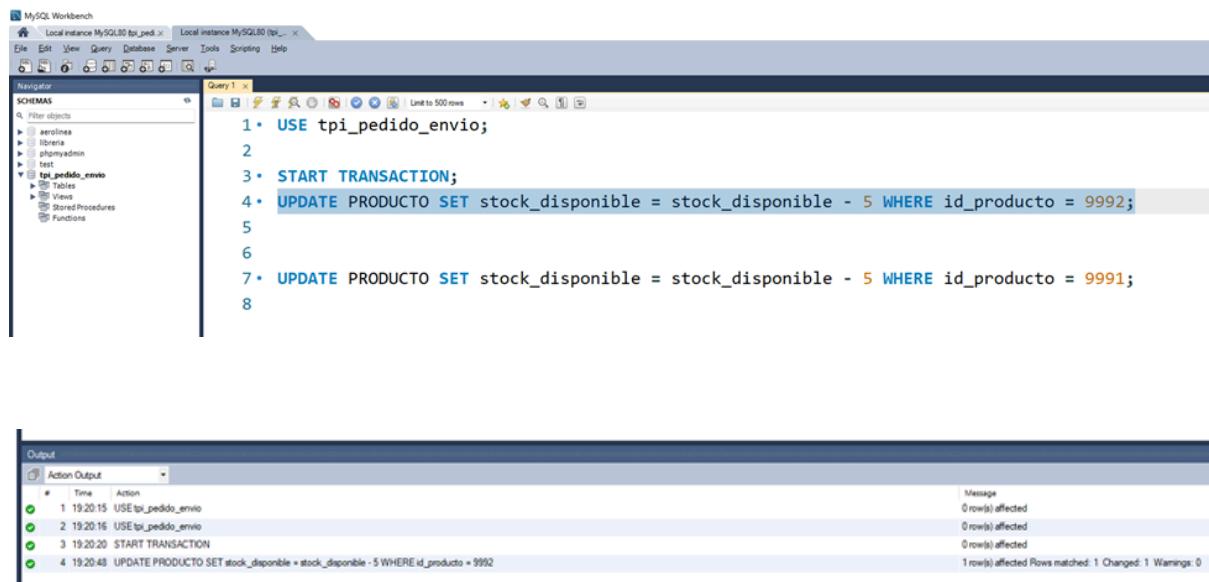
```
1 • USE tpi_pedido_envio;
2
3 • START TRANSACTION;
4 • UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE id_producto = 9991;
5
6
7 • UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE id_producto = 9992;
```

Action Output:

#	Time	Action	Message
1	19:17:36	USE tpi_pedido_envio	0 row(s) affected
2	19:17:36	INSERT INTO PRODUCTO (id_producto, producto_nombre, producto_codigo, precio_unitario, stock_disponible) VALUES (991, 'Producto A', 'TEST-A', 100.00, 50), (992, 'Producto B', 'TEST-B', 200.00, 50)	0 row(s) affected Records: 2 Duplicates: 0 Warnings: 0
3	19:19:47	USE tpi_pedido_envio	0 row(s) affected
4	19:19:52	START TRANSACTION	0 row(s) affected
5	19:19:56	UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE id_producto = 9991	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

SESION 2

```
USE tpi_pedido_envio;
START TRANSACTION;
UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5
WHERE id_producto = 9992;
```



The screenshot shows the MySQL Workbench interface with a script editor and an output window.

Script Editor:

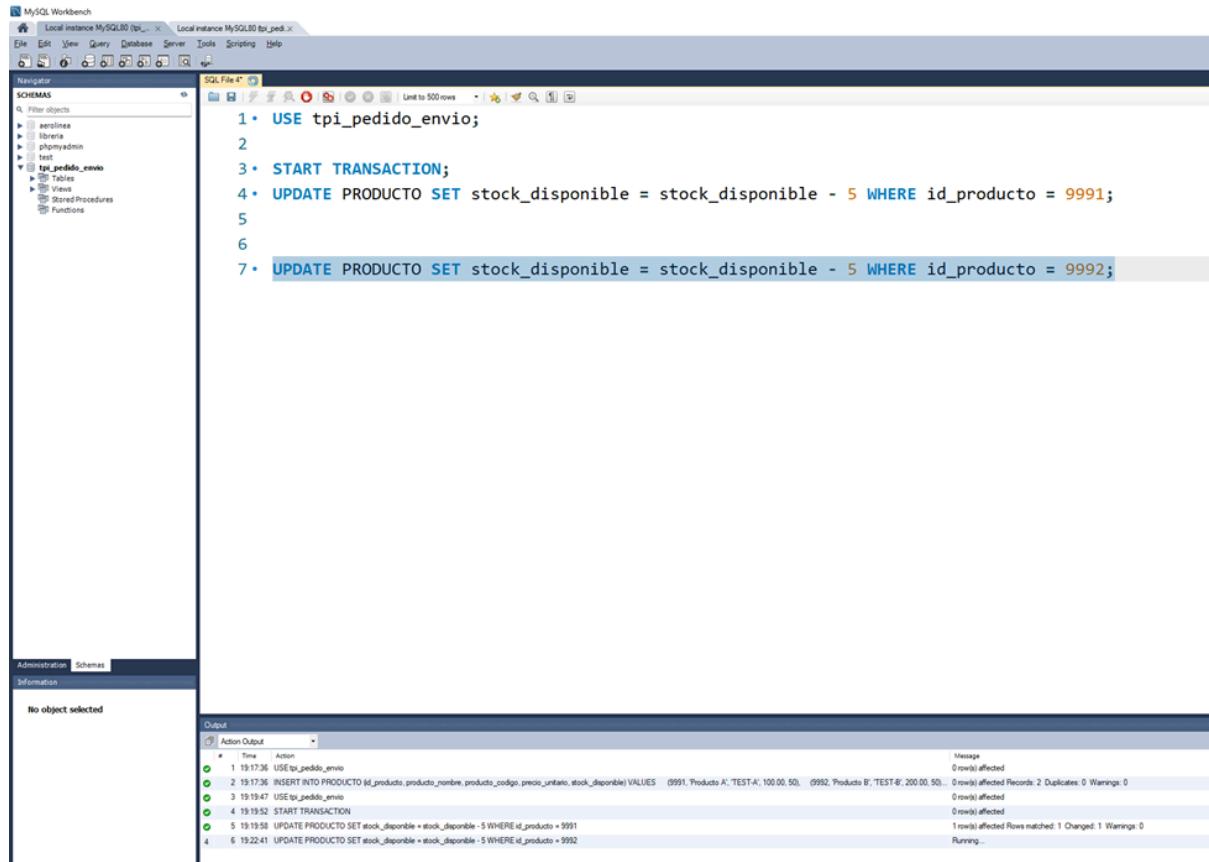
```
1 • USE tpi_pedido_envio;
2
3 • START TRANSACTION;
4 • UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE id_producto = 9992;
5
6
7 • UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE id_producto = 9991;
8
```

Action Output:

#	Time	Action	Message
1	19:20:15	USE tpi_pedido_envio	0 row(s) affected
2	19:20:16	USE tpi_pedido_envio	0 row(s) affected
3	19:20:20	START TRANSACTION	0 row(s) affected
4	19:20:48	UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE id_producto = 9992	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

SESION 1

```
UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE
id_producto = 9992;
```



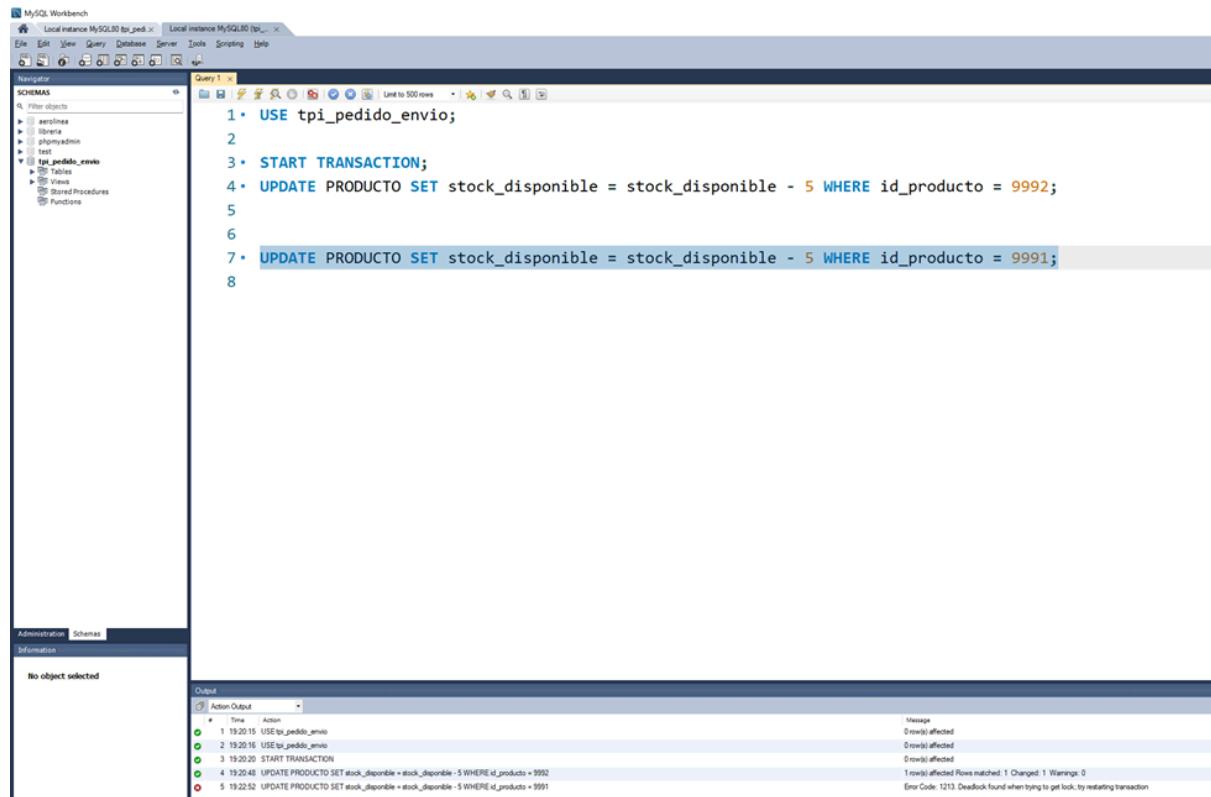
The screenshot shows the MySQL Workbench interface with a transaction log. The log details the following steps:

- 1. USE tpi_pedido_envio;
- 2.
- 3. START TRANSACTION;
- 4. UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE id_producto = 9991;
- 5.
- 6.
- 7. UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE id_producto = 9992;

The log also shows the execution of an INSERT INTO statement and the start of a transaction. The final update step (step 7) is highlighted in blue, indicating it is the current operation being run.

SESION 2

```
UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE
id_producto = 9991;
```



The screenshot shows the MySQL Workbench interface with a transaction log. The log contains the following entries:

```

1 • USE tpi_pedido_envio;
2
3 • START TRANSACTION;
4 • UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE id_producto = 9992;
5
6
7 • UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE id_producto = 9991;
8

```

The log shows two failed UPDATE statements (lines 4 and 7) due to a deadlock. The output pane displays the following error message:

```

Message
Error(s) affected
Error(s) affected
Error(s) affected
Rows(s) affected Rows matched: 1 Changed: 1 Warnings: 0
Error Code: 1213. Deadlock found when trying to get lock; try restarting transaction

```

SESION 1

SHOW ENGINE INNODB STATUS;

```
instance MySQL80 tpi_ped.x
Tools Scripting Help
SQL File 4" x
1 • USE tpi_pedido_envio;
2
3 • START TRANSACTION;
4 • UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE id_producto = 9991;
5
6
7 • UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE id_producto = 9992;
8
9 • SHOW ENGINE INNODB STATUS;
```

Type	Name	Status
InnoDB	=====...	

WE PULL BACK. THANKS FOR TURNING ON ME!

Type	Name
InnoDB	=====...

TRANSACTIONS

```
Trx id counter: 1187
Purge list length: 23
History list length: 23
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 283184295412632, not started
    0 rows inserted, 0 rows updated, 0 rows deleted
---TRANSACTION 1184, ACTIVE 314 sec
    3 lock struct(s), heap size 1128, 2 row lock(s), undo log entries 2
MySQL thread id 30, OS thread handle 206276, query id 927 localhost:1 root
SHOW ENGINE INNODB STATUS
```

FILE I/O

```
1 • USE tpi_li
2
3 • START TR
4 • UPDATE PI
5
6
7 • UPDATE PI
8
9 • SHOW ENG.
```

id_producto = 9991;

id_producto = 9992;

INSERT BUFFER AND ADAPTIVE HASH INDEX

```
-----
```

read size 1, free list len 342, seg size 244, 9878 merges

merged operations:

- insert 616373, delete mark 0, delete 0
- discarded operations:
- insert 0, delete 0, update 0
- Hash table size 4441, node heap has 98 buffer(s)
- Hash table size 4441, node heap has 1 buffer(s)
- Hash table size 4441, node heap has 1 buffer(s)
- Hash table size 4441, node heap has 1 buffer(s)
- Hash table size 4441, node heap has 1 buffer(s)
- Hash table size 4441, node heap has 1 buffer(s)
- Hash table size 4441, node heap has 1 buffer(s)
- Hash table size 4441, node heap has 1 buffer(s)
- Hash table size 4441, node heap has 1 buffer(s)
- Hash table size 4441, node heap has 1 buffer(s)
- 0.00 hash searches/s, 0.00 non-hash searches/s

LOG

```
Log sequence number 223449441
Log flushed up to 223449441
Pages flushed up to 223449441
Last checkpoint at 223449432
Pending log flushes, 0 pending chkpt writes
1684 log file(s), 0.00 log i/o/s/second
```

BUFFER POOL AND MEMORY

```
Total large memory allocated 33554432
Dictionary memory allocated 57592
Buffer pool size 1003
Free buffers 642
Total pages 256
Old database pages 0
Modified db pages 0
Percent of dirty pages(RU & free pages): 0.000
Dirty page percent: 75.000
Pending writes 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 58792, not young 5866716
0.00 youngs/s, 0.00 non-youngs/s
Pages written 72836, created 691, written 45951
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU list length: 0, group_LRU len: 0
I/O sum[0]:cur[0], uncip sum[0]:cur[0]
```

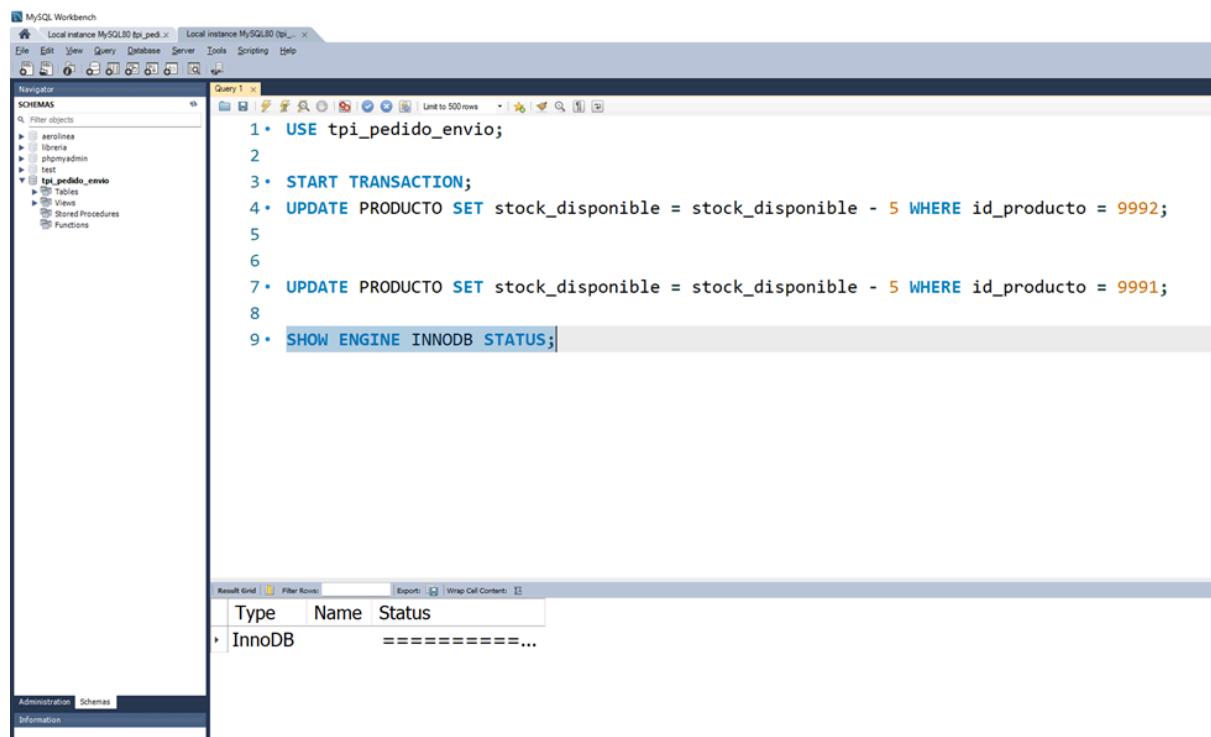
ROW OPERATIONS

```
0 queries inside InnoDB, 0 queries in queue
0 read views open inside InnoDB
Process ID=10532, Main thread ID=22644, state: sleeping
Number of rows inserted 0, updated 0, deleted 6, read 0
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
Number of system rows inserted 0, updated 0, deleted 0, read 0
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
.....
```

END OF INNODB MONITOR OUTPUT

SESION 2

```
SHOW ENGINE INNODB STATUS;
```



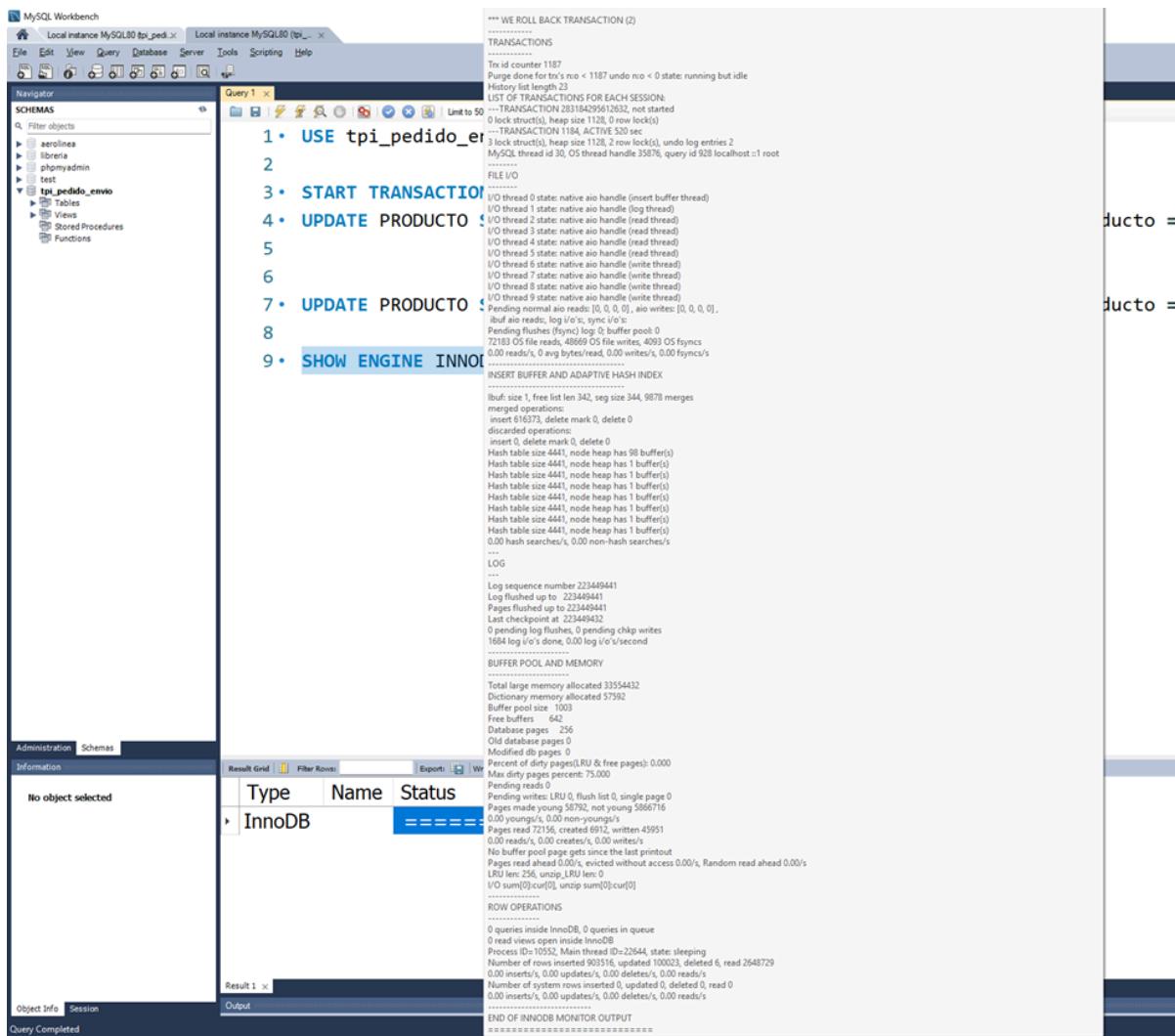
The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' section, the 'tpi_pedido_envio' database is selected. The main area contains a query editor window titled 'Query 1'. The query text is:

```
1 • USE tpi_pedido_envio;
2
3 • START TRANSACTION;
4 • UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE id_producto = 9992;
5
6
7 • UPDATE PRODUCTO SET stock_disponible = stock_disponible - 5 WHERE id_producto = 9991;
8
9 • SHOW ENGINE INNODB STATUS;
```

Below the query editor is a 'Result Grid' window. It has three columns: 'Type', 'Name', and 'Status'. The data is as follows:

Type	Name	Status
InnoDB		=====...

At the bottom of the interface, there is a navigation bar with tabs: 'Administration', 'Schemas' (which is currently active), and 'Information'.



```

*** WE ROLL BACK TRANSACTION (2)
=====
TRANSACTIONS
=====
Trx id counter 1187
Purge done for trx's no < 1187 undo no < 0 state: running but idle
History list length 23
--TRANSACTION 283184295612632, not started
0 lock struct(s), 0 lock(s), 0 row lock(s), 0 wait(s)
--TRANSACTION 1184, ACTIVE 520 sec
3 lock struct(s), heap size 1128, 2 row lock(s), undo log entries 2
MySQL thread id 30, OS thread handle 35876, query id 928 localhost ::1 root
=====
FILE I/O
=====
I/O thread 0 state: native aio handle (insert buffer thread)
I/O thread 1 state: native aio handle (log thread)
I/O thread 2 state: native aio handle (read thread)
I/O thread 3 state: native aio handle (read thread)
I/O thread 4 state: native aio handle (read thread)
I/O thread 5 state: native aio handle (read thread)
I/O thread 6 state: native aio handle (write thread)
I/O thread 7 state: native aio handle (write thread)
I/O thread 8 state: native aio handle (write thread)
I/O thread 9 state: native aio handle (write thread)
Pending normal aio reads: [0, 0, 0], aio writes: [0, 0, 0],
ibuf aio reads: log i/o's, sync i/o's:
Pending flushes (fsync): log 0; buffer pool: 0
72183 OS file reads, 40569 OS file writes, 4093 OS fymcs
0.00 read/s, 0 avg bytes/read, 0.00 writes/s, 0.00 fymcs/s
=====
INSERT BUFFER AND ADAPTIVE HASH INDEX
=====
Ibuf: size 1, free list len 342, seg size 344, 9878 merges
merged operations:
insert 616773, delete mark 0, delete 0
discarded operations:
insert 0, delete mark 0, delete 0
Hash table size 4441, node heap has 98 buffer(s)
Hash table size 4441, node heap has 1 buffer(s)
Hash table size 4441, node heap has 1 buffer(s)
Hash table size 4441, node heap has 1 buffer(s)
Hash table size 4441, node heap has 1 buffer(s)
Hash table size 4441, node heap has 1 buffer(s)
Hash table size 4441, node heap has 1 buffer(s)
Hash table size 4441, node heap has 1 buffer(s)
0.00 hash searched/s, 0.00 non-hash searches/s
=====
LOG
=====
Log sequence number 223449441
Log flushed up to 223449441
Pages flushed up to 223449441
Last checkpoint at 223449432
0 pending log flushes, 0 pending chkpt writes
1684 log i/o's done, 0.00 log i/o's/second
=====
BUFFER POOL AND MEMORY
=====
Total large memory allocated 33554432
Dictionary memory allocated 57952
Buffer pool size 1003
Free buffers 642
Database pages 256
Old database pages 0
Modified database pages 0
Percent dirty pages(0RU & free pages): 0.00
Pending reads 0
Pending writes LRU 0, flush list 0, single page 0
Pages made young 0, not young 586776
0.00 read/s, 0.00 non-read/s
Pages read 2715, created 6912, written 45951
0.00 read/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00%, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 256, unsp.LRU len: 0
U/O sum(0)cur(0) unsp.sum(0)cur(0)
=====
ROW OPERATIONS
=====
0 updates inside InnoDB, 0 queries in queue
0 read views open inside InnoDB
Process ID=10532, Main thread ID=22544, state: sleeping
Number of rows inserted 903516, updated 100023, deleted 6, read 2648729
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
Number of sys rows inserted 0, updated 0, deleted 0, read 0
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
=====
END OF INNODB MONITOR OUTPUT
=====
```

Para esta prueba preparamos dos productos de prueba con id_producto 9991 y 9992 y abrimos dos sesiones simultáneas en MySQL Workbench.

En la primera sesión bloqueamos el producto 9991, y en la segunda bloqueamos el 9992.

Como siguiente paso, intentamos que cada sesión acceda al producto bloqueado por la otra, creando un ciclo de espera.

MySQL detecta esta situación y aborta una de las transacciones con el Error Code: 1213. Deadlock found when trying to get lock; try restarting transaction.

Esto demuestra que el motor tiene un mecanismo de detección de deadlocks que previene bloqueos indefinidos

READ COMMITTED

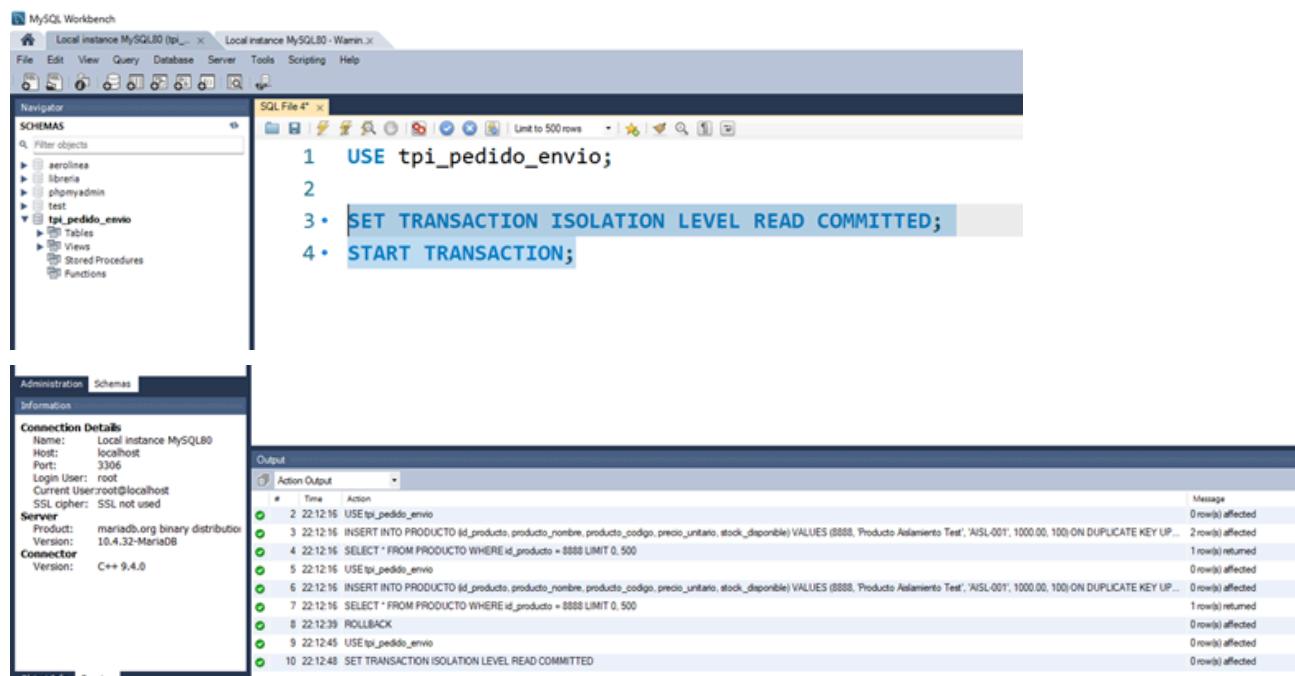
Preparamos los datos sobre los cuales trabajaremos.

```
USE tpi_pedido_envio;
INSERT INTO PRODUCTO (id_producto, producto_nombre, producto_codigo,
precio_unitario, stock_disponible)
VALUES (8888, 'Producto Aislamiento Test', 'AISL-001', 1000.00, 100)
ON DUPLICATE KEY UPDATE stock_disponible = 50;
-- Verificar
SELECT * FROM PRODUCTO WHERE id_producto = 8888;
```

SESIÓN 1

```
USE tpi_pedido_envio;
      SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
      SELECT stock_disponible AS 'Lectura 1'
FROM PRODUCTO
      WHERE id_producto = 8888;
```

Resultado esperado Lectura 1 = 50.



The screenshot shows the MySQL Workbench interface. In the top navigation bar, the database is set to 'Local instance MySQL80 (tpi_pedido_envio)'. The main window has a 'SQL File 4*' tab open. The SQL code in the editor is:

```
1 USE tpi_pedido_envio;
2
3 • SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
4 • START TRANSACTION;
```

In the bottom right corner, the 'Output' pane displays the execution log:

#	Time	Action	Message
1	22:12:15	USE tpi_pedido_envio	0 row(s) affected
2	22:12:16	INSERT INTO PRODUCTO (id_producto, producto_nombre, producto_codigo, precio_unitario, stock_disponible) VALUES (8888, 'Producto Aislamiento Test', 'AISL-001', 1000.00, 100) ON DUPLICATE KEY UP...	2 row(s) affected
3	22:12:16	SELECT * FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned
4	22:12:16	USE tpi_pedido_envio	0 row(s) affected
5	22:12:16	INSERT INTO PRODUCTO (id_producto, producto_nombre, producto_codigo, precio_unitario, stock_disponible) VALUES (8888, 'Producto Aislamiento Test', 'AISL-001', 1000.00, 100) ON DUPLICATE KEY UP...	0 row(s) affected
6	22:12:16	SELECT * FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned
7	22:12:16	ROLLBACK	0 row(s) affected
8	22:12:45	USE tpi_pedido_envio	0 row(s) affected
9	22:12:48	SET TRANSACTION ISOLATION LEVEL READ COMMITTED	0 row(s) affected

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- Local instance MySQL80 (tpi...)
- Local instance MySQL80 - Wmnn...
- Filter objects
- airlines
- base
- phpmyadmin
- test
- tpi_pedido_envio
- Tables
- Views
- Stored Procedures
- Functions

SQL File 4*:

```

1 USE tpi_pedido_envio;
2
3 • SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
4 • START TRANSACTION;
5
6 • SELECT stock_disponible AS 'Lectura 1'
7 FROM PRODUCTO
8 WHERE id_producto = 8888;
  
```

Result Grid | Filter Rows: Export: Wrap Cell Content: E

Lectura
1
50

Administration Schemas Information

Connection Details

Name: Local instance MySQL80
Host: localhost
Port: 3306
Login User: root
Current User:root@localhost
SSL cipher: SSL not used

Server

Product: mariadb.org binary distribution
Version: 10.4.32-MariaDB

Connector

C++ 9.4.0

PRODUCTO 9*:

Action Output

Time	Action	Message
3 22:12:16	INSERT INTO PRODUCTO (id_producto, producto_nombre, producto_codigo, precio_unitario, stock_disponible) VALUES (8888, 'Producto Alimento Test', 'AISL-001', 1000.00, 100) ON DUPLICATE KEY UPDATE	2 row(s) affected
4 22:12:16	SELECT * FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned
5 22:12:16	USE tpi_pedido_envio	0 row(s) affected
6 22:12:16	INSERT INTO PRODUCTO (id_producto, producto_nombre, producto_codigo, precio_unitario, stock_disponible) VALUES (8888, 'Producto Alimento Test', 'AISL-001', 1000.00, 100) ON DUPLICATE KEY UPDATE	0 row(s) affected
7 22:12:16	SELECT * FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned
8 22:12:39	ROLLBACK	0 row(s) affected
9 22:12:45	USE tpi_pedido_envio	0 row(s) affected
10 22:12:48	SET TRANSACTION ISOLATION LEVEL READ COMMITTED	0 row(s) affected
11 22:13:07	SELECT stock_disponible AS 'Lectura 1' FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned

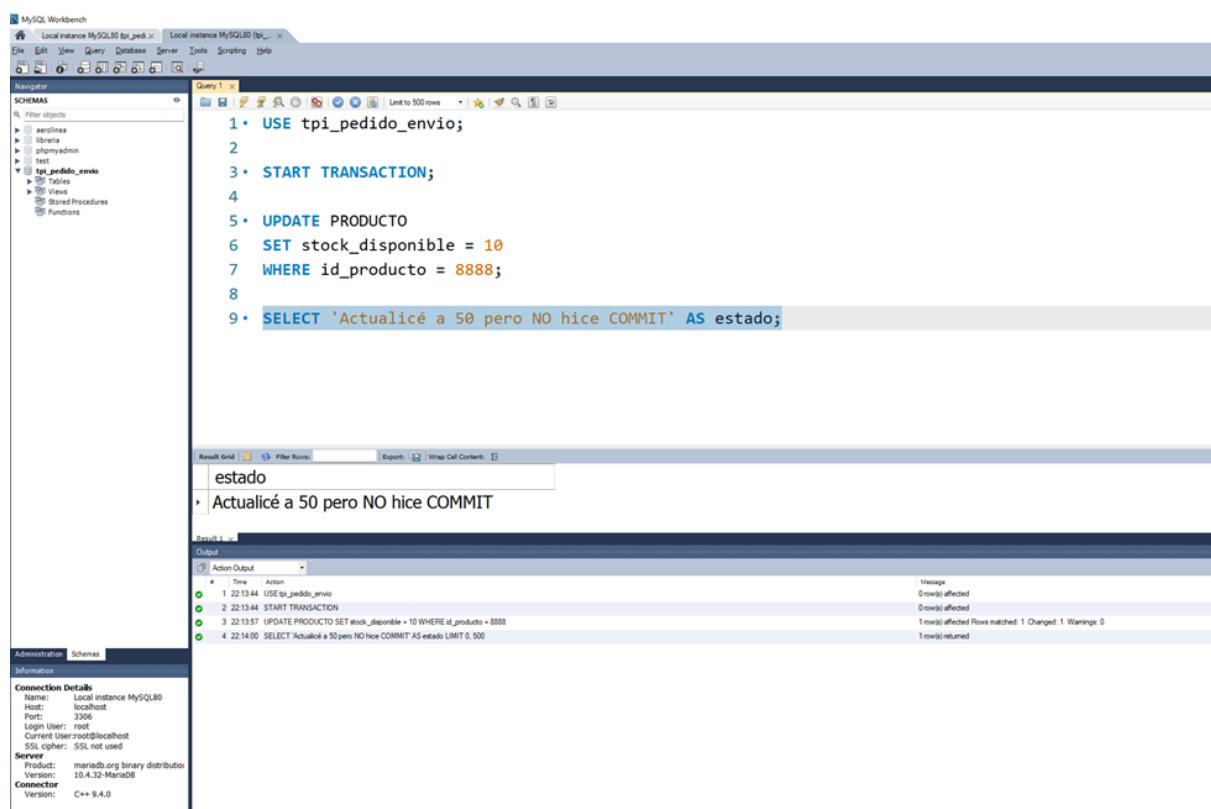
Segunda lectura

SESIÓN 2

```
USE tpi_pedido_envio;
START TRANSACTION;
UPDATE PRODUCTO
SET stock_disponible = 10
WHERE id_producto = 8888;
SELECT 'Actualicé a 50 pero NO hice COMMIT' AS estado;
```

Se espera un mensaje indicando que se actualizó (pero la sesión todavía no hizo COMMIT).

Cabe destacar que el comentario quedó desactualizado con el valor 50 en lugar de 10.



The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database structure with the schema `tpi_pedido_envio` expanded, revealing tables like `articulos`, `libreria`, `logins`, `productos`, and `test`.
- Query Editor (Query 1):** Contains the SQL code shown in the text block above.
- Result Grid (Result 1):** Shows the output of the query `SELECT 'Actualicé a 50 pero NO hice COMMIT' AS estado;`. It displays a single row with the value `Actualicé a 50 pero NO hice COMMIT`.
- Output (Action Output):** Shows the log of actions with their times and descriptions. The log includes:

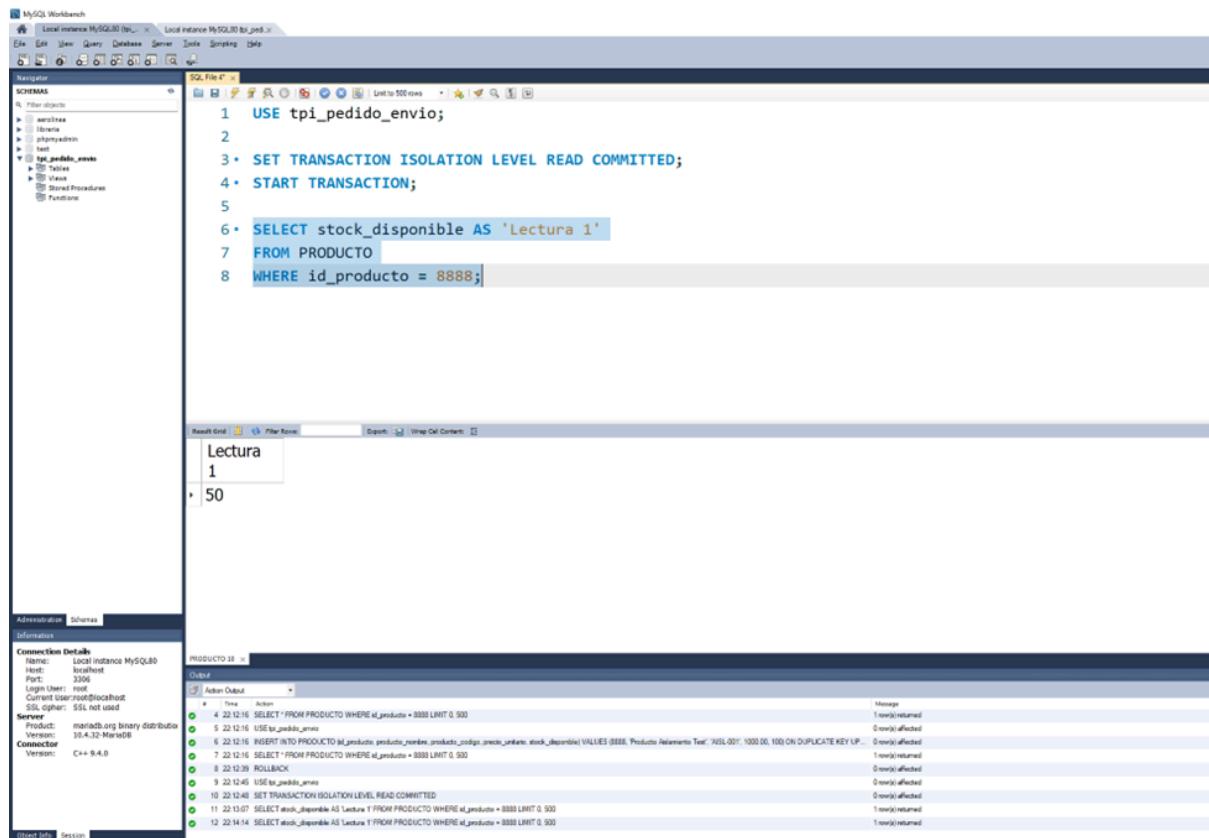
#	Time	Action	Message
1	22:13:44	USE tpi_pedido_envio	0 row(s) affected
2	22:13:44	START TRANSACTION	0 row(s) affected
3	22:13:57	UPDATE PRODUCTO SET stock_disponible = 10 WHERE id_producto = 8888	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
4	22:14:00	SELECT 'Actualicé a 50 pero NO hice COMMIT' AS estado LIMIT 0,500	1 row(s) returned
- Connection Details:** Provides information about the current connection, including the host (localhost), port (3306), and user (root@localhost).

SESIÓN 1

Lectura 2 (antes del COMMIT de sesión 2):

```
SELECT stock_disponible AS 'Lectura 1'
FROM PRODUCTO
WHERE id_producto = 8888;
```

La sesión 1 no ve el cambio no confirmado.



The screenshot displays the MySQL Workbench interface with two sessions:

- Session 1 (Top Window):**
 - Query Editor: Contains the SQL code for a SELECT statement:


```
1 USE tpi_pedido_envio;
2
3 • SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
4 • START TRANSACTION;
5
6 • SELECT stock_disponible AS 'Lectura 1'
7 FROM PRODUCTO
8 WHERE id_producto = 8888;
```
 - Result Grid: Shows the output of the query:

Lectura
1
50
- Session 2 (Bottom Window):**
 - Query Editor: Shows the execution of the transaction:


```
4 22:12:16 SELECT * FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500
5 22:12:16 USE tpi_pedido_envio
6 22:12:16 INSERT INTO PRODUCTO (id_producto, producto_nombre, producto_codigo, precio_unitario, stock_disponible) VALUES (8888, 'Producto Aislamiento Test', 'ASL-001', 1000.00, 100) ON DUPLICATE KEY UP...
7 22:12:16 SELECT * FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500
8 22:12:39 ROLLBACK
9 22:12:45 USE tpi_pedido_envio
10 22:12:45 SET TRANSACTION ISOLATION LEVEL READ COMMITTED
11 22:13:07 SELECT stock_disponible AS Lectura 1 FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500
12 22:14:14 SELECT stock_disponible AS Lectura 1 FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500
```
 - Result Grid: Shows the output of the final SELECT statement from Session 2:

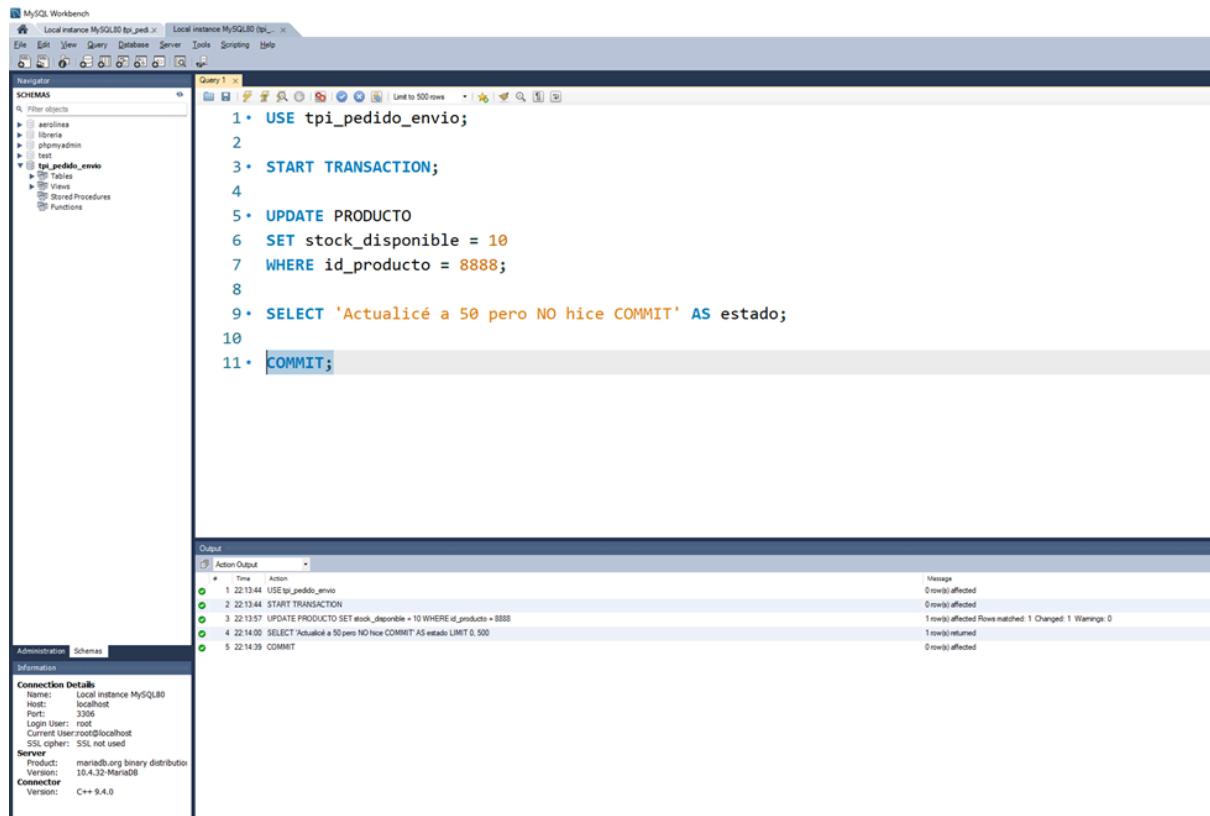
Lectura
1
50

The Connection Details pane on the left shows the connection information for both sessions.

SESIÓN 2

`COMMIT;`

Ahora el cambio está persistiendo.



The screenshot shows the MySQL Workbench interface with a query editor and an output window. The query editor contains the following SQL code:

```

1 • USE tpi_pedido_envio;
2
3 • START TRANSACTION;
4
5 • UPDATE PRODUCTO
6   SET stock_disponible = 10
7   WHERE id_producto = 8888;
8
9 • SELECT 'Actualicé a 50 pero NO hice COMMIT' AS estado;
10
11• COMMIT;
  
```

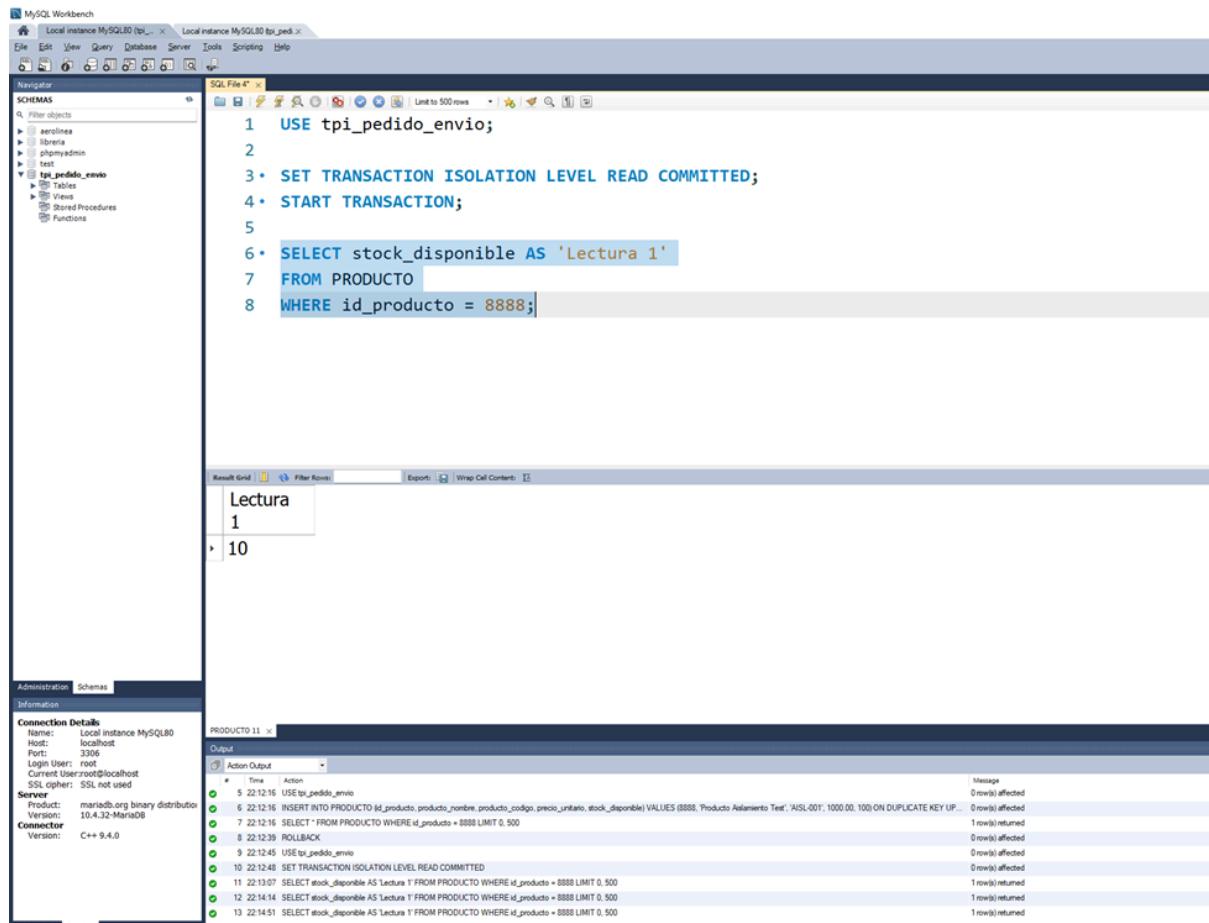
The output window displays the execution log:

#	Time	Action	Message
1	22:13:44	USE tpi_pedido_envio	0 rows/affected
2	22:13:44	START TRANSACTION	0 rows/affected
3	22:13:57	UPDATE PRODUCTO SET stock_disponible = 10 WHERE id_producto = 8888	1 rows/affected Rows matched: 1 Changed: 1 Warnings: 0
4	22:14:00	SELECT 'Actualicé a 50 pero NO hice COMMIT' AS estado LIMIT 0, 500	1 rows/returned
5	22:14:39	COMMIT	0 rows/affected

SESIÓN 1

```
SELECT stock_disponible AS 'Lectura 1'
FROM PRODUCTO
WHERE id_producto = 8888;
```

En **READ COMMITTED** la sesión ve commits hechos por otras sesiones.



The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database structure with the schema `tpi_pedido_envio` selected.
- SQL File 4*:** Contains the following SQL code:


```
1 USE tpi_pedido_envio;
2
3• SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
4• START TRANSACTION;
5
6• SELECT stock_disponible AS 'Lectura 1'
7 FROM PRODUCTO
8 WHERE id_producto = 8888;
```
- Result Grid:** Displays the result of the query:

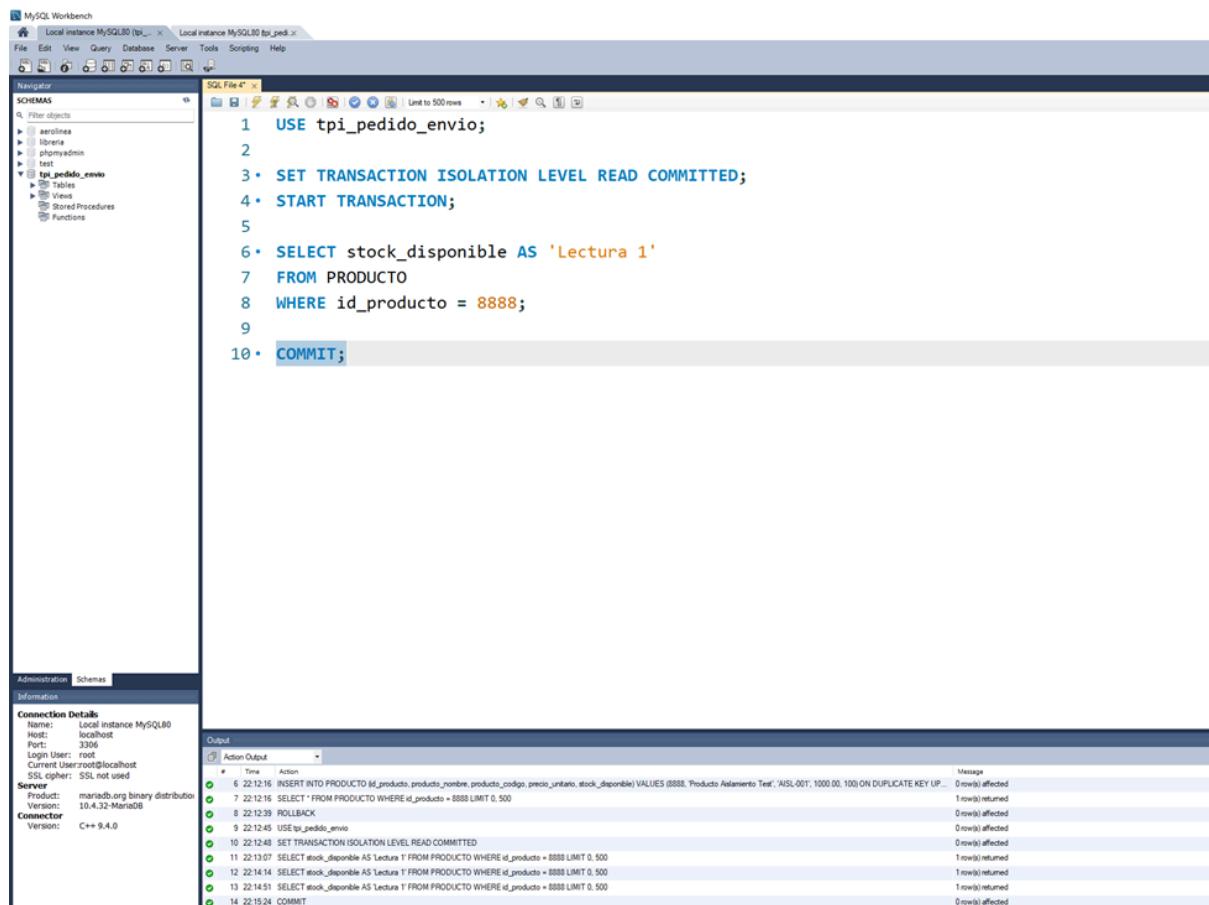
Lectura
1
10
- PRODUCTO 11..** Shows the transaction log with the following entries:

Action	Time	Message
5 22:12:16 USE tpi_pedido_envio		0 row(s) affected
6 22:12:16 INSERT INTO PRODUCTO (id_producto, producto_nombre, producto_codigo, precio_unitario, stock_disponible) VALUES (8888, 'Producto Alimento Test', 'AISL-001', 1000.00, 100) ON DUPLICATE KEY UPDATE stock_disponible = 100		0 row(s) affected
7 22:12:16 SELECT * FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500		1 row(s) returned
8 22:12:39 ROLLBACK		0 row(s) affected
9 22:12:45 USE tpi_pedido_envio		0 row(s) affected
10 22:12:48 SET TRANSACTION ISOLATION LEVEL READ COMMITTED		0 row(s) affected
11 22:13:07 SELECT stock_disponible AS 'Lectura 1' FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500		1 row(s) returned
12 22:14:14 SELECT stock_disponible AS 'Lectura 1' FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500		1 row(s) returned
13 22:14:51 SELECT stock_disponible AS 'Lectura 1' FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500		1 row(s) returned

SESIÓN 1

COMMIT;

(si en SESIÓN 1 no hay cambios, COMMIT cierra la transacción).



The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database schema with the **tpi_pedido_envio** database selected.
- SQL File 4:** Contains the following SQL code:


```

1 USE tpi_pedido_envio;
2
3 • SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
4 • START TRANSACTION;
5
6 • SELECT stock_disponible AS 'Lectura 1'
7 FROM PRODUCTO
8 WHERE id_producto = 8888;
9
10 • COMMIT;
```
- Connection Details:**
 - Name: Local Instance MySQL80
 - Host: localhost
 - Port: 3306
 - Login User: root
 - Current User: root@localhost
 - SSL cipher: SSL not used
- Server:**
 - Product: mariadb.org binary distribution
 - Version: 10.4.32-MariaDB
 - Connector: C++ 9.4.0
- Output:** Displays the transaction log with the following entries:

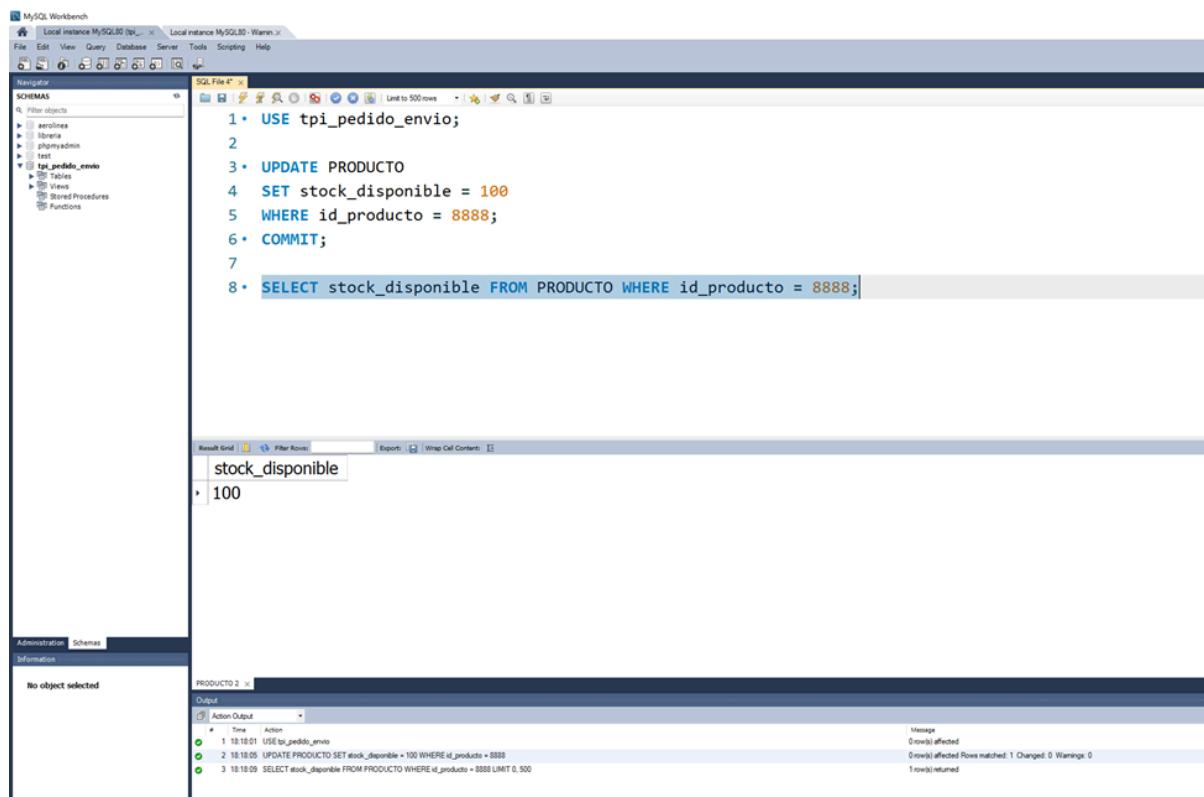
#	Time	Action	Message
6	22:12:16	INSERT INTO PRODUCTO (id_producto, producto_nombre, producto_codigo, precio_unitario, stock_disponible) VALUES (8888, 'Producto Asimiento Test', 'AISL-001', 1000.00, 100) ON DUPLICATE KEY UPDATE stock_disponible = 100	0 row(s) affected
7	22:12:16	SELECT * FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned
8	22:12:39	ROLLBACK	0 row(s) affected
9	22:12:45	USE tpi_pedido_envio	0 row(s) affected
10	22:12:48	SET TRANSACTION ISOLATION LEVEL READ COMMITTED	0 row(s) affected
11	22:13:07	SELECT stock_disponible AS 'Lectura 1' FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned
12	22:14:14	SELECT stock_disponible AS 'Lectura 1' FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned
13	22:14:51	SELECT stock_disponible AS 'Lectura 1' FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned
14	22:15:24	COMMIT	0 row(s) affected

REPEATABLE READ

Previo a la ejecución restauramos los valores de stock

SESIÓN 1

```
UPDATE PRODUCTO
    SET stock_disponible = 100
    WHERE id_producto = 8888;
COMMIT;
SELECT stock_disponible FROM PRODUCTO WHERE id_producto = 8888;
```



The screenshot shows the MySQL Workbench interface. In the SQL Editor, the following code is run:

```
USE tpi_pedido_envio;
2
3 • UPDATE PRODUCTO
4   SET stock_disponible = 100
5   WHERE id_producto = 8888;
6 • COMMIT;
7
8 • SELECT stock_disponible FROM PRODUCTO WHERE id_producto = 8888;
```

The Results Grid displays the result of the SELECT statement:

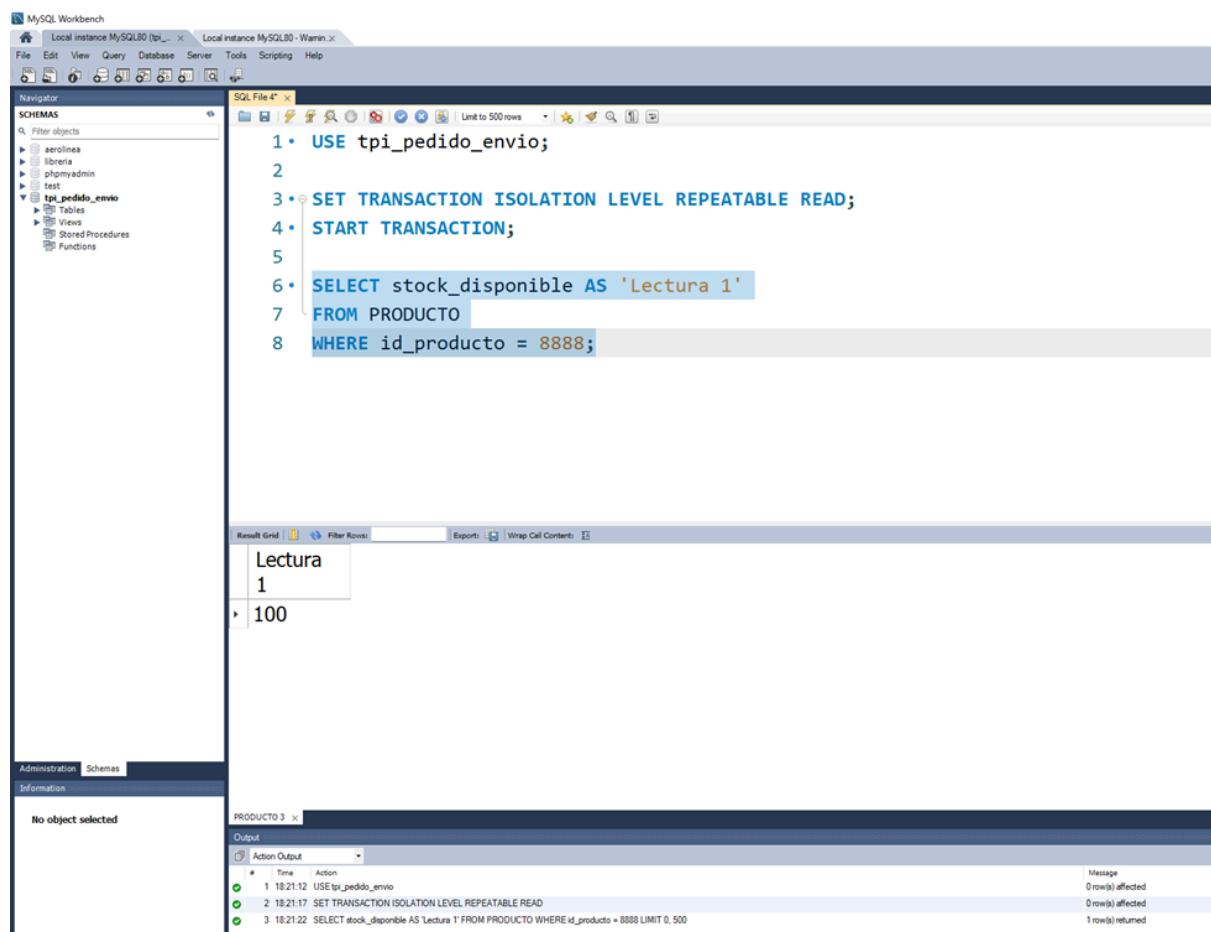
stock_disponible
100

The Output pane shows the transaction log:

#	Time	Action	Message
1	18:01:01	USE tpi_pedido_envio	0 rows affected
2	18:01:05	UPDATE PRODUCTO SET stock_disponible = 100 WHERE id_producto = 8888	0 rows affected Rows matched: 1 Changed: 0 Warnings: 0
3	18:01:09	SELECT stock_disponible FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 rows returned

```
USE tpi_pedido_envio;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;
```

Resultado esperado es stock = 100



The screenshot shows the MySQL Workbench interface with the following details:

- SQL File 4:** Contains the following SQL code:


```

1 • USE tpi_pedido_envio;
2
3 • SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
4 • START TRANSACTION;
5
6 • SELECT stock_disponible AS 'Lectura 1'
7 FROM PRODUCTO
8 WHERE id_producto = 8888;
```
- Result Grid:** Displays the result of the query:

Lectura
1
100
- Output:** Shows the log of actions taken during the transaction:

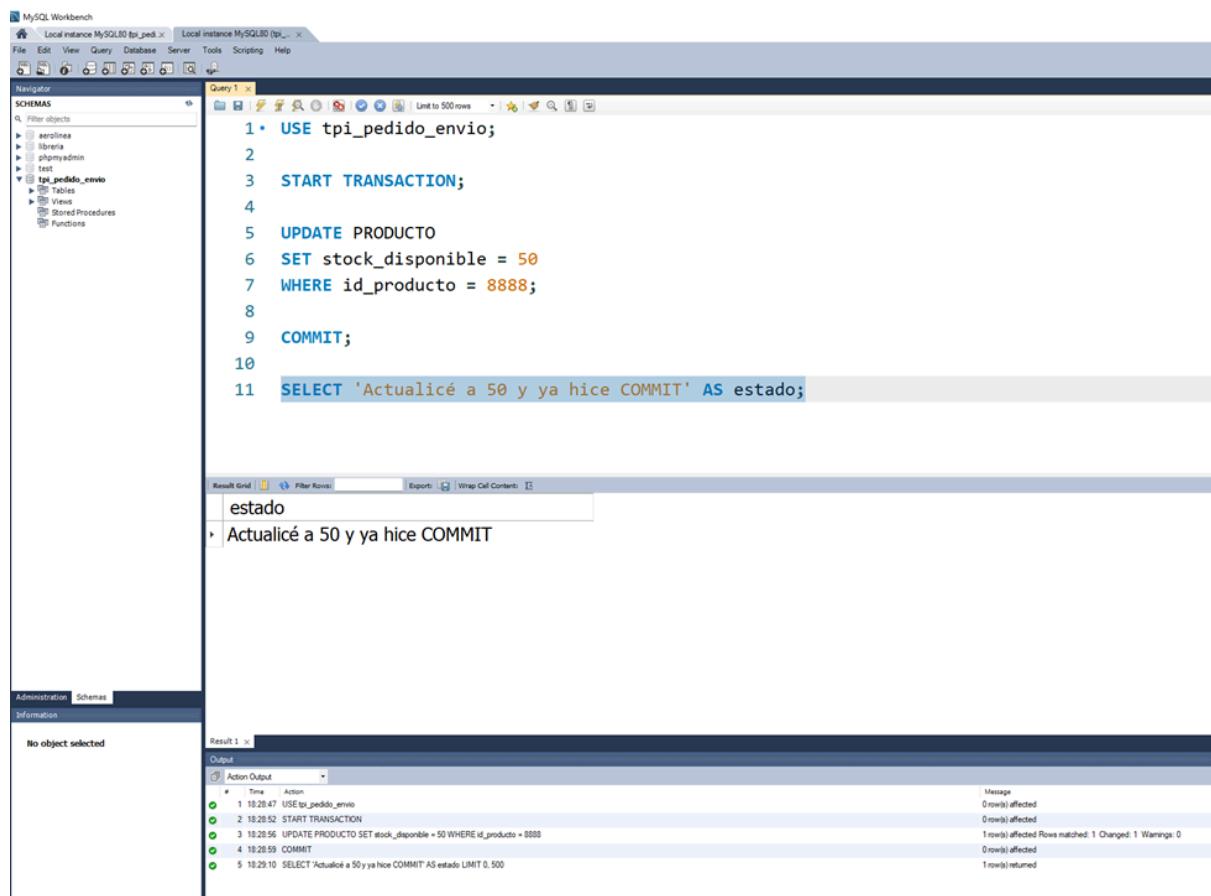
Action Output	Time	Action	Message
1	18:21:12	USE tpi_pedido_envio	0 row(s) affected
2	18:21:17	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ	0 row(s) affected
3	18:21:22	SELECT stock_disponible AS Lectura 1 FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned

SESIÓN 2

```

USE tpi_pedido_envio;
START TRANSACTION;
  UPDATE PRODUCTO
    SET stock_disponible = 50
      WHERE id_producto = 8888;
COMMIT;
SELECT 'Actualicé a 50 y ya hice COMMIT' AS estado;
  
```

Confirmamos que se hizo COMMIT y ahora la tabla contiene 50.



The screenshot shows the MySQL Workbench interface with two panes. The left pane displays the Navigator and Schemas sections, showing the database structure. The right pane has a 'Query 1' tab containing the SQL code for the transaction, and a 'Result Grid' tab showing the output of the SELECT statement. Below these is an 'Output' section showing the transaction log.

Query 1:

```

1 • USE tpi_pedido_envio;
2
3 START TRANSACTION;
4
5 UPDATE PRODUCTO
6   SET stock_disponible = 50
7   WHERE id_producto = 8888;
8
9 COMMIT;
10
11 SELECT 'Actualicé a 50 y ya hice COMMIT' AS estado;
  
```

Result Grid:

estado	
Actualicé a 50 y ya hice COMMIT	

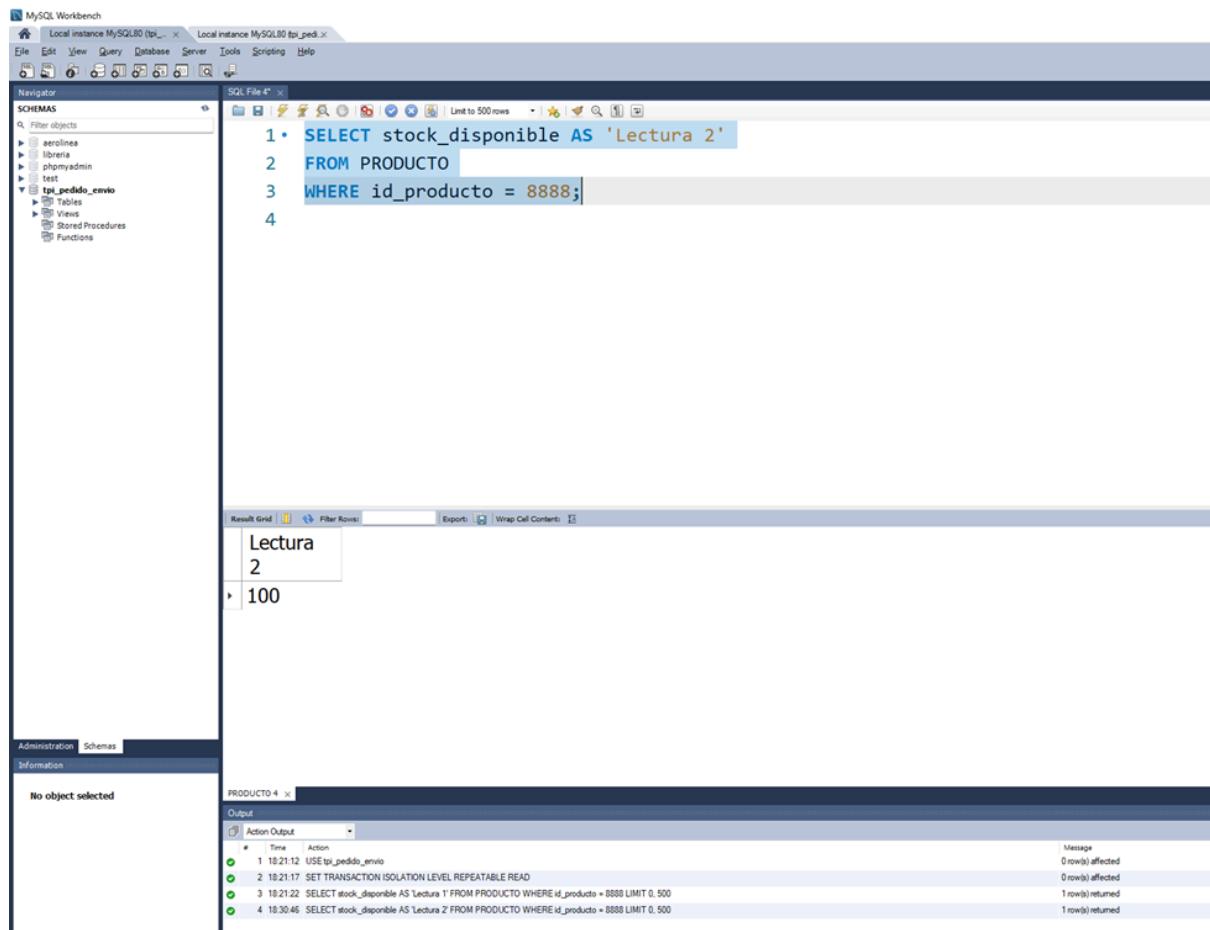
Output:

#	Time	Action	Message
1	10:28:47	USE tpi_pedido_envio	0 row(s) affected
2	10:28:52	START TRANSACTION	0 row(s) affected
3	10:28:56	UPDATE PRODUCTO SET stock_disponible = 50 WHERE id_producto = 8888	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
4	10:28:59	COMMIT	0 row(s) affected
5	10:29:10	SELECT 'Actualicé a 50 y ya hice COMMIT' AS estado LIMIT 0, 500	1 row(s) returned

SESIÓN 1

```
SELECT stock_disponible AS 'Lectura 2'
FROM PRODUCTO
WHERE id_producto = 8888;
```

El resultado esperado es 100 ya que con **REPEATABLE READ** la transacción mantiene la vista consistente que tenía al inicio; y todavía no ve el commit.



The screenshot shows the MySQL Workbench interface with the following details:

- SQL Editor:** Contains the following SQL code:


```
1 • SELECT stock_disponible AS 'Lectura 2'
2   FROM PRODUCTO
3 WHERE id_producto = 8888;
4
```
- Result Grid:** Displays the results of the query:

Lectura
2
100
- Output Window:** Shows the transaction log with the following entries:

#	Time	Action	Message
1	18:21:12	USE `tp1_pedido_envio`	0 row(s) affected
2	18:21:17	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ	0 row(s) affected
3	18:21:22	SELECT stock_disponible AS 'Lectura 1' FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned
4	18:30:46	SELECT stock_disponible AS 'Lectura 2' FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned

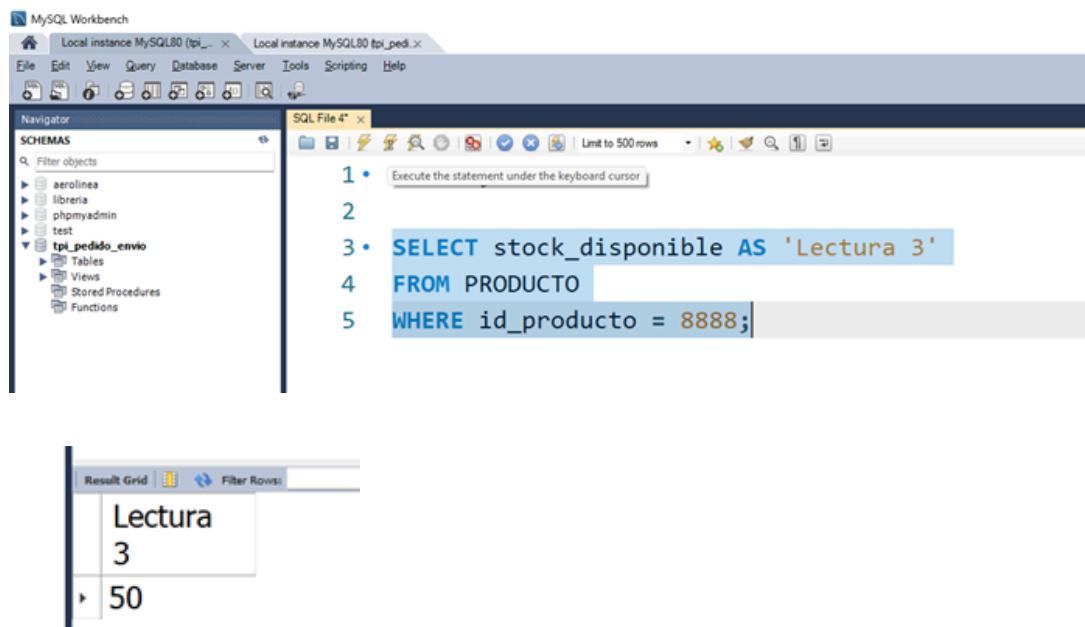
SESIÓN 1

```
COMMIT;
```

Al terminar la transacción ya se puede hacer una nueva lectura de la tabla.

```
SELECT stock_disponible AS 'Lectura 3'
FROM PRODUCTO
WHERE id_producto = 8888;
```

Ahora esperamos visualizar 50 como valor de stock, ya que después de cerrar la transacción, la sesión muestra el valor committed por SESIÓN 2.



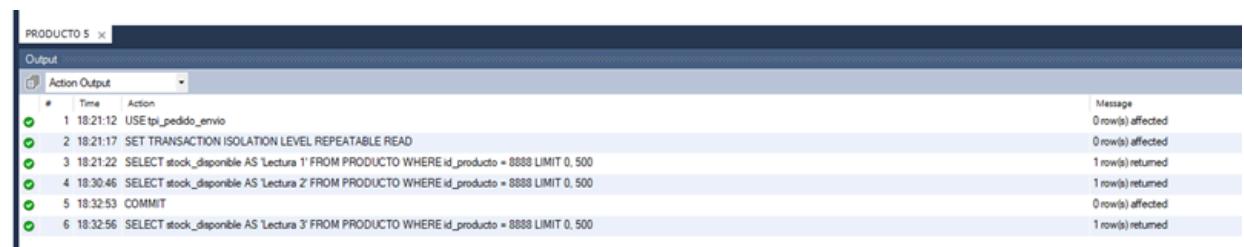
The screenshot shows the MySQL Workbench interface. In the Navigator pane, under the SCHEMAS section, the 'tpi_pedido_envio' schema is selected. In the SQL File 4* editor, the following query is written:

```
1 • Execute the statement under the keyboard cursor
2
3 • SELECT stock_disponible AS 'Lectura 3'
4 FROM PRODUCTO
5 WHERE id_producto = 8888;
```

In the Result Grid, the output is displayed in a table:

Lectura 3
50

Tenemos la lectura con la sentencia y los logs de la misma consulta



The screenshot shows the MySQL Workbench log window titled 'PRODUCTO 5'. It displays the following log entries:

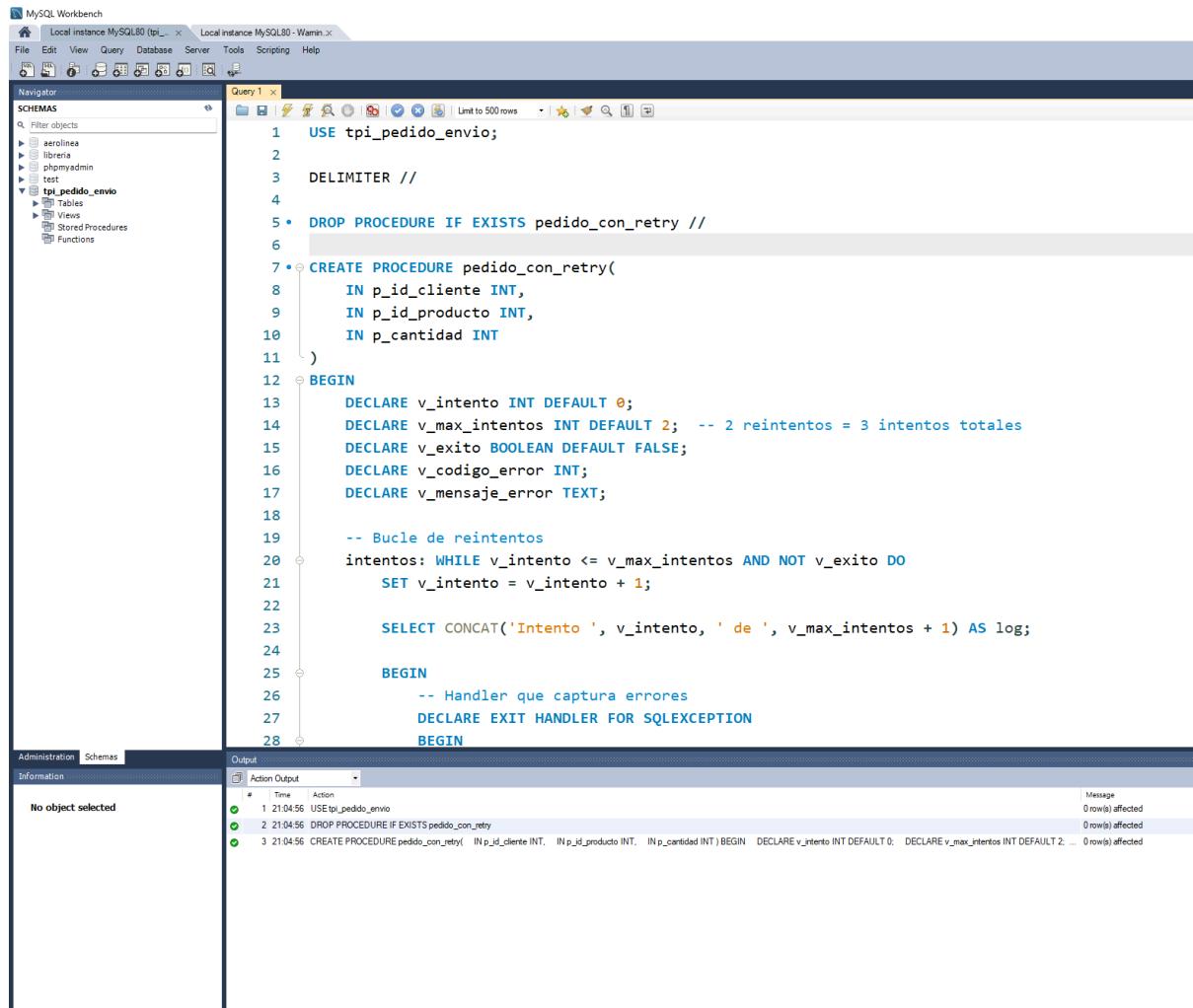
Action	Time	Action	Message
1	18:21:12	USE tpi_pedido_envio	0 row(s) affected
2	18:21:17	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ	0 row(s) affected
3	18:21:22	SELECT stock_disponible AS 'Lectura 1' FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned
4	18:30:46	SELECT stock_disponible AS 'Lectura 2' FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned
5	18:32:53	COMMIT	0 row(s) affected
6	18:32:56	SELECT stock_disponible AS 'Lectura 3' FROM PRODUCTO WHERE id_producto = 8888 LIMIT 0, 500	1 row(s) returned

Al comparar **READ COMMITTED** con **REPEATABLE READ**, notamos que con el primero una misma consulta devuelve resultados diferentes dentro de la misma transacción si en otra sesión se realizan cambios, lo que puede causar inconsistencias, por otro lado **REPEATABLE READ**, nos garantiza que los **SELECT** de la consulta muestra solo un estado consistente de los datos durante la transacción, eliminando el problema de inconsistencias encontrados en **READ COMMITTED**.

Retry ante deadlock

Manejar un error 1213/40001

Usamos una sentencia de SQL para generar y manejar el deadlock.



The screenshot shows the MySQL Workbench interface with a query editor and an output window. The query editor contains the following SQL code:

```

1 USE tpi_pedido_envio;
2
3 DELIMITER //
4
5 • DROP PROCEDURE IF EXISTS pedido_con_retry //
6
7 • CREATE PROCEDURE pedido_con_retry(
8     IN p_id_cliente INT,
9     IN p_id_producto INT,
10    IN p_cantidad INT
11 )
12 BEGIN
13     DECLARE v_intento INT DEFAULT 0;
14     DECLARE v_max_intentos INT DEFAULT 2; -- 2 reintentos = 3 intentos totales
15     DECLARE v_exito BOOLEAN DEFAULT FALSE;
16     DECLARE v_codigo_error INT;
17     DECLARE v_mensaje_error TEXT;
18
19     -- Bucle de reintentos
20     intentos: WHILE v_intento <= v_max_intentos AND NOT v_exito DO
21         SET v_intento = v_intento + 1;
22
23         SELECT CONCAT('Intento ', v_intento, ' de ', v_max_intentos + 1) AS log;
24
25     BEGIN
26         -- Handler que captura errores
27         DECLARE EXIT HANDLER FOR SQLEXCEPTION
28         BEGIN

```

The output window shows the results of the execution:

#	Time	Action	Message
1	21:04:56	USE tpi_pedido_envio	0 row(s) affected
2	21:04:56	DROP PROCEDURE IF EXISTS pedido_con_retry	0 row(s) affected
3	21:04:56	CREATE PROCEDURE pedido_con_retry(IN p_id_cliente INT, IN p_id_producto INT, IN p_cantidad INT) BEGIN DECLARE v_intento INT DEFAULT 0; DECLARE v_max_intentos INT DEFAULT 2; ... 0 row(s) affected	

```

USE tpi_pedido_envio;DELIMITER //DROP PROCEDUREIF EXISTS
pedido_con_retry // 
CREATE PROCEDURE pedido_con_retry( IN p_id_cliente int,
                                    IN p_id_producto int,
                                    IN p_cantidad int )
BEGIN
  DECLARE v_intento      int DEFAULT 0;
  declare v_max_intentos int DEFAULT 2; -- 2 reintentos = 3 intentos
totales
  declare v_exito         boolean DEFAULT false;
  declare v_codigo_error int;
  declare v_mensaje_error text;
  -- Bucle de reintentos
INTENTOS:
  WHILE v_intento <= v_max_intentos
  AND
  NOT v_exito do
    SET v_intento = v_intento + 1;
    select concat('Intento ', v_intento, ' de ', v_max_intentos + 1)
AS log;

begin
  -- Handler que captura errores
  DECLARE EXIT handler FOR sqlexception
  BEGIN
    get diagnostics condition 1 v_codigo_error = mysql_errno,
v_mensaje_error = message_text;
    select concat('ERROR detectado - Código: ', v_codigo_error, ' '
- Mensaje: ', v_mensaje_error) AS log;

    -- Si es DEADLOCK (error 1213)
    if v_codigo_error = 1213 THEN
      IF v_intento <= v_max_intentos THEN
        SELECT 'DEADLOCK detectado. Aplicando backoff de 1 segundo...' 
AS log;

        do sleep(1); -- BACKOFF
        else
          SELECT 'FALLO: Deadlock persistente después de reintentos' AS
resultado;
      end
      IF
      else
        -- Otro error: no reintentar
        SELECT concat('ERROR NO RECUPERABLE: ', v_mensaje_error) AS
resultado;
    end
  end
END
  
```

```

      set v_intento = v_max_intentos + 1;
    end
    IF :
end;
-- ===== TRANSACCIÓN =====START TRANSACTION,INSERT INTO pedido
(
      id_cliente,
      fecha_pedido,
      estado_pedido,
      total
)
VALUES
(
      p_id_cliente,
      Now(),
      'Pendiente',
      0
) ;SET @nuevo_pedido = Last_insert_id();INSERT INTO
pedido_producto
(
      id_pedido,
      id_producto,
      cantidad,
      precio_unitario,
      subtotal
)
SELECT @nuevo_pedido,
      p_id_producto,
      p_cantidad,
      precio_unitario,
      precio_unitario * p_cantidad
FROM producto
WHERE id_producto = p_id_producto;UPDATE pedido
SET total =
(
      SELECT Sum(subtotal)
      FROM pedido_producto
      WHERE id_pedido = @nuevo_pedido)
WHERE id_pedido = @nuevo_pedido;COMMIT;SET v_exito = true;SELECT
Concat('EXITO: Pedido ', @nuevo_pedido, ' registrado en intento ',
v_intento, ' - COMMIT ejecutado')
AS resultado;LEAVE intentos;END;
END WHILE intentos;IF NOT v_exito then
SELECT 'Operación fallida después de todos los intentos' AS
resumen;ENDIF;END // 
delimiter ;

```

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- Filter objects
- aerolinea
- libreta
- phpadmin
- test
- tpi_pedido_envio**
- Tables
- Views
- Stored Procedures
- Functions

Query 1

```
1 CALL pedido_con_retry(1, 1, 5);
```

Result Grid | Filter Rows | Export | Wrap Cell Content:

log

Intento 1 de 3

Result 51 × Result 52

Administration Schemas Information

No object selected

Action Output

#	Time	Action	Message
1	21:04:56	USE tpi_pedido_envio	0 row(s) affected
2	21:04:56	DROP PROCEDURE IF EXISTS pedido_con_retry	0 row(s) affected
3	21:04:56	CREATE PROCEDURE pedido_con_retry(IN p_id_cliente INT, IN p_id_producto INT, IN p_cantidad INT) BEGIN DECLARE v_intento INT DEFAULT 0; DECLARE v_max_intentos INT DEFAULT 2; WHILE v_intento < v_max_intentos AND p_cantidad > 0 DO SET p_cantidad = p_cantidad - 1; SET v_intento = v_intento + 1; END WHILE; END;	0 row(s) affected
4	21:05:57	CALL pedido_con_retry(1, 1, 5)	1 row(s) returned
5	21:05:57	CALL pedido_con_retry(1, 1, 5)	1 row(s) returned

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- Filter objects
- aerolinea
- libreta
- phpadmin
- test
- tpi_pedido_envio**
- Tables
- Views
- Stored Procedures
- Functions

Query 1

```
1 CALL pedido_con_retry(1, 1, 5);
```

Result Grid | Filter Rows | Export | Wrap Cell Content:

resultado

EXITO: Pedido 100018 registrado en intento 1 - COMMIT ejecutado

Result 51 × Result 52 ×

Administration Schemas Information

No object selected

Action Output

#	Time	Action	Message
1	21:04:56	USE tpi_pedido_envio	0 row(s) affected
2	21:04:56	DROP PROCEDURE IF EXISTS pedido_con_retry	0 row(s) affected
3	21:04:56	CREATE PROCEDURE pedido_con_retry(IN p_id_cliente INT, IN p_id_producto INT, IN p_cantidad INT) BEGIN DECLARE v_intento INT DEFAULT 0; DECLARE v_max_intentos INT DEFAULT 2; WHILE v_intento < v_max_intentos AND p_cantidad > 0 DO SET p_cantidad = p_cantidad - 1; SET v_intento = v_intento + 1; END WHILE; END;	0 row(s) affected
4	21:05:57	CALL pedido_con_retry(1, 1, 5)	1 row(s) returned
5	21:05:57	CALL pedido_con_retry(1, 1, 5)	1 row(s) returned

```
CALL pedido_con_retry(1, 1, 5);
```

MySQL Workbench

Local instance MySQL80 (tpi_pedido_envio) Local instance MySQL80 - Warnings

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS Filter objects

- seorlinea
- libreria
- phpmyadmin
- tpi_pedido_envio
- Tables
- Views
- Stored Procedures
- Functions

Query1 x

```
1 CALL pedido_con_retry(999999, 1, 5);
```

Result Grid Filter Rows Export Wrap Cell Content

log

Intento 1 de 3

Administration Schemas Information

No object selected

Action Output

#	Time	Action	Message
1	21:04:56	USE tpi_pedido_envio	0 row(s) affected
2	21:04:56	DROP PROCEDURE IF EXISTS pedido_con_retry	0 row(s) affected
3	21:04:56	CREATE PROCEDURE pedido_con_retry(IN p_id_cliente INT, IN p_id_producto INT, IN p_caridad INT) BEGIN DECLARE v_intento INT DEFAULT 0; DECLARE v_max_intento INT DEFAULT 2;...	0 row(s) affected
4	21:05:57	CALL pedido_con_retry(1, 1, 5)	1 row(s) returned
5	21:05:57	CALL pedido_con_retry(1, 1, 5)	1 row(s) returned
6	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned
7	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned
8	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned
9	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned

MySQL Workbench

Local instance MySQL80 (tpi_pedido_envio) Local instance MySQL80 - Warnings

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS Filter objects

- seorlinea
- libreria
- phpmyadmin
- tpi_pedido_envio
- Tables
- Views
- Stored Procedures
- Functions

Query1 x

```
1 CALL pedido_con_retry(999999, 1, 5);
```

Result Grid Filter Rows Export Wrap Cell Content

log

• ERROR detectado - Código: 1452 - Mensaje: Cannot add or update a child row: a foreign key constraint fails ('tpi_pedido_envio'.`pedi...

Administration Schemas Information

No object selected

Action Output

#	Time	Action	Message
1	21:04:56	USE tpi_pedido_envio	0 row(s) affected
2	21:04:56	DROP PROCEDURE IF EXISTS pedido_con_retry	0 row(s) affected
3	21:04:56	CREATE PROCEDURE pedido_con_retry(IN p_id_cliente INT, IN p_id_producto INT, IN p_caridad INT) BEGIN DECLARE v_intento INT DEFAULT 0; DECLARE v_max_intento INT DEFAULT 2;...	0 row(s) affected
4	21:05:57	CALL pedido_con_retry(1, 1, 5)	1 row(s) returned
5	21:05:57	CALL pedido_con_retry(1, 1, 5)	1 row(s) returned
6	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned
7	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned
8	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned
9	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned

MySQL Workbench

Local instance MySQL80 (tpi_pedido_envio) Local instance MySQL80 - Wamin

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- Filter objects
- Tables
- Views
- Stored Procedures
- Functions

Query 1 x

```
1 CALL pedido_con_retry(999999, 1, 5);
```

Result Grid Filter Rows Export Wrap Cell Content

resultado

- ERROR NO RECUPERABLE: Cannot add or update a child row: a foreign key constraint fails ('tpi_pedido_envio`.'pedido', CONSTRAINT `fk_pedido_cliente` FOREIGN KEY(`id_cliente`) REFERENCES `cliente`(`id_cliente`))

Administration Schemas Information

No object selected

Action Output

#	Time	Action	Message
1	21:04:56	USE tpi_pedido_envio	0 row(s) affected
2	21:04:56	DROP PROCEDURE IF EXISTS pedido_con_retry	0 row(s) affected
3	21:04:56	CREATE PROCEDURE pedido_con_retry(IN p_id_cliente INT, IN p_id_producto INT, IN p_cantidad INT) BEGIN DECLARE v_intento INT DEFAULT 0; DECLARE v_max_intento INT DEFAULT 2; WHILE v_intento <= v_max_intento DO IF p_cantidad > 0 THEN INSERT INTO pedido SET id_cliente = p_id_cliente, id_producto = p_id_producto, cantidad = p_cantidad; SET p_cantidad = p_cantidad - 1; END IF; SET v_intento = v_intento + 1; END WHILE; END;	0 row(s) affected
4	21:05:57	CALL pedido_con_retry(1, 5)	1 row(s) returned
5	21:05:57	CALL pedido_con_retry(1, 5)	1 row(s) returned
6	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned
7	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned
8	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned
9	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned

MySQL Workbench

Local instance MySQL80 (tpi_pedido_envio) Local instance MySQL80 - Wamin

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- Filter objects
- Tables
- Views
- Stored Procedures
- Functions

Query 1 x

```
1 CALL pedido_con_retry(999999, 1, 5);
```

Result Grid Filter Rows Export Wrap Cell Content

resumen

- Operación fallida después de todos los intentos

Administration Schemas Information

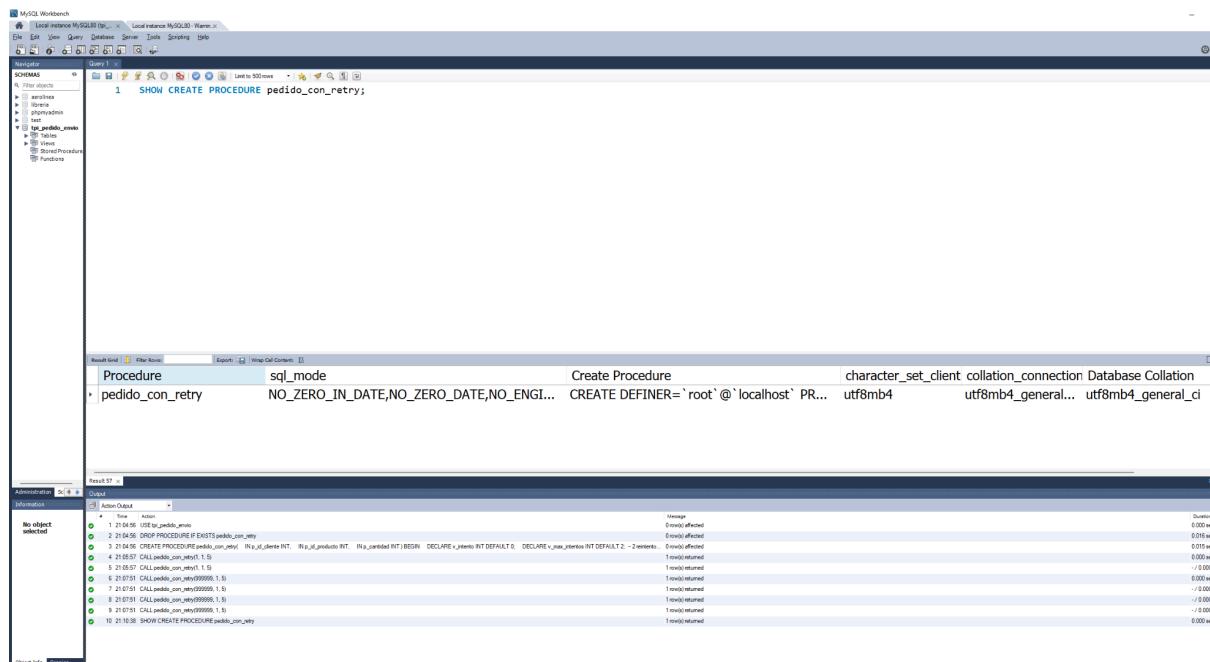
No object selected

Action Output

#	Time	Action	Message
1	21:04:56	USE tpi_pedido_envio	0 row(s) affected
2	21:04:56	DROP PROCEDURE IF EXISTS pedido_con_retry	0 row(s) affected
3	21:04:56	CREATE PROCEDURE pedido_con_retry(IN p_id_cliente INT, IN p_id_producto INT, IN p_cantidad INT) BEGIN DECLARE v_intento INT DEFAULT 0; DECLARE v_max_intento INT DEFAULT 2; WHILE v_intento <= v_max_intento DO IF p_cantidad > 0 THEN INSERT INTO pedido SET id_cliente = p_id_cliente, id_producto = p_id_producto, cantidad = p_cantidad; SET p_cantidad = p_cantidad - 1; END IF; SET v_intento = v_intento + 1; END WHILE; END;	0 row(s) affected
4	21:05:57	CALL pedido_con_retry(1, 5)	1 row(s) returned
5	21:05:57	CALL pedido_con_retry(1, 5)	1 row(s) returned
6	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned
7	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned
8	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned
9	21:07:51	CALL pedido_con_retry(999999, 1, 5)	1 row(s) returned

Object Info Session

```
CALL pedido_con_retry(999999, 1, 5);
```



The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The current schema is 'aristoteles'.
- Query Editor:** The query 'SHOW CREATE PROCEDURE pedido_con_retry;' is run, and the results are displayed in the Result Grid.
- Result Grid:**

Procedure	sql_mode	Create Procedure	character_set_client	collation_connection	Database	Collation
pedido_con_retry	NO_ZERO_IN_DATE,NO_ZERO_DATE,NO_ENGI...	CREATE DEFINER='root'@'localhost' PR...	utf8mb4	utf8mb4_general...	utf8mb4	utf8mb4_general_ci
- Action Output:** This section shows the execution history of the procedure creation and execution. It includes log entries for each step, such as creating the procedure, dropping it, and then executing it 10 times with parameters (999999, 1, 5). Each execution entry includes a timestamp, message, rows affected, and duration.

```
SHOW CREATE PROCEDURE pedido_con_retry;
```

Implementamos un procedimiento almacenado con estrategia de retry ante **deadlocks**.

El procedimiento intenta ejecutar una transacción (START TRANSACTION/COMMIT/ROLLBACK) que registra un pedido completo.

Si ocurre algún error durante la ejecución, el **EXIT HANDLER** captura el código de error mediante **GET DIAGNOSTICS**.

Si el código es **1213** (deadlock), el procedimiento espera 1 segundo (backoff) mediante DO SLEEP(1) y reintenta la operación hasta 2 veces adicionales (3 intentos totales).

Si el error no es un deadlock (por ejemplo, violación de foreign key código 1452), considera que es un error no recuperable y no reintenta.

Durante las pruebas verificamos que:

- Las transacciones exitosas ejecutan **COMMIT** correctamente
- Los errores activan **ROLLBACK** automático mediante el handler
- La lógica detecta correctamente el **código 1213** para aplicar retry
- Otros errores no causan reintentos innecesarios

Este mecanismo hace que el sistema sea resiliente ante deadlocks momentáneos sin requerir intervención manual, ya que generalmente el reintento después del backoff tiene éxito sólo cuando la transacción que causó el deadlock libera sus recursos.

Referencias: Anexo Ref 3.

Anexo

Uso de IA por etapa:

Etapa 1:

<https://gemini.google.com/share/e49bebe8b2c3>
<https://chatgpt.com/share/68f80bb6-ff78-800a-8414-3873c5ebacb0>

Etapa 2:

<https://gemini.google.com/share/9a185045ec52>

Etapa 3:

<https://gemini.google.com/share/18abfc9094a>

Referencias

Ref 1:

<https://www.youtube.com/watch?v=wD2OxnqmKgc>
<https://www.youtube.com/watch?v=kcuMQnGly68>

Ref 0.1:

https://www.w3schools.com/sql/sql_foreignkey.asp
https://www.w3schools.com/sql/sql_check.asp
https://www.w3schools.com/sql/sql_unique.asp
<https://stackoverflow.com/questions/6720050/foreign-key-constraints-when-to-use-on-update-and-on-delete>

Ref 2:

<https://dev.mysql.com/doc/refman/8.4/en/create-view.html>

Ref 3:

<https://learn.microsoft.com/es-es/sql/relational-databases/sql-server-deadlocks-guide?view=sql-server-ver17>
<https://www.w3resource.com/sql-exercises/sql-query-to-handle-deadlocks-using-error-handling-techniques.php>
<https://www.w3resource.com/mysql-exercises/mysql-query-to-handle-a-deadlock-situation-by-retrying-the-transaction.php>
<https://lextonr.wordpress.com/2012/04/14/retry-execution-even-on-deadlock/>

Enlace al video

https://drive.google.com/file/d/1OkAiYYVa08oyJBqNy2emxUWhPeskh8pW/view?usp=drive_link