

Universidad Tecnológica Nacional

Tecnicatura Universitaria en Programación a Distancia



Título del trabajo: **“Trabajo Final Integrador (TFI) - Bases de Datos I”**

Alumnos: Eric Suárez Dubs, ericgodzilladubs@gmail.com - Comisión 13

Gonzalo Vega, gonzaarg03@gmail.com - Comisión 11

Federico Iacono, iaconofede@gmail.com - Comisión 6

Mateo Serafini, matuserafini@gmail.com - Comisión 5

Materia: **Bases de datos I**

[Video explicativo](#)

Profesor Coordinador: **Oscar Londero**

Fecha de Entrega: **23/10/2025**

Índice

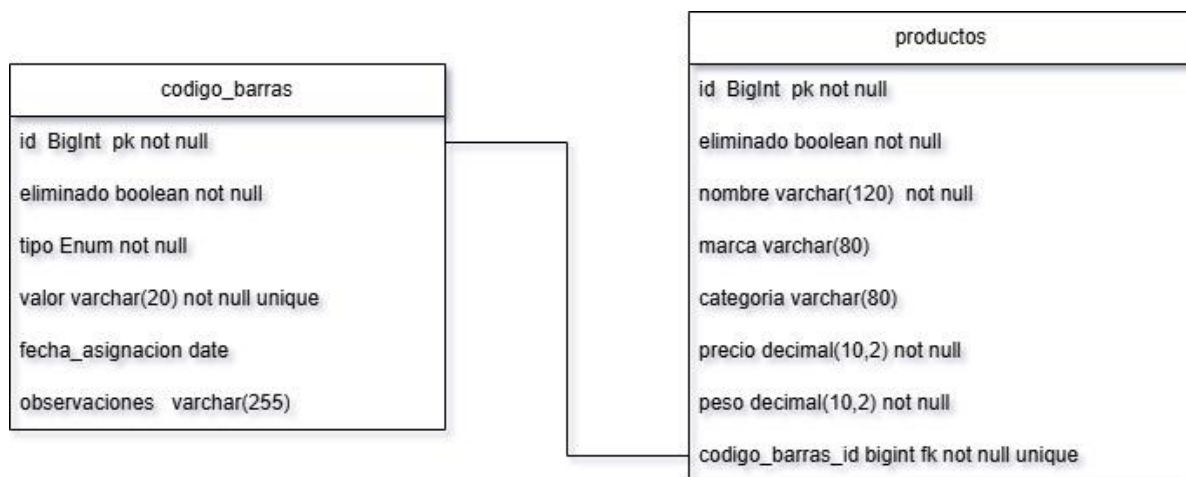
1. Introducción.
2. Etapa 1 – Modelado y Definición de Constraints
3. Etapa 2 – Generación y carga de datos masivos con SQL puro
4. Etapa 3 – Consultas Avanzadas y Reportes
5. Etapa 4 – Seguridad e Integridad
6. Etapa 5 – Concurrency y Transacciones
7. Conclusión.
8. Anexo.

1. Introducción

En el desarrollo de esta actividad hemos optado por implementar un modelo especializado de base de datos para la gestión de códigos de barras de productos, utilizando como herramienta fundamental MySQL Workbench y aplicando de manera integral los conocimientos teórico-prácticos adquiridos durante el desarrollo de nuestra formación en base a las herramientas adquiridas en el transcurso de nuestro aprendizaje. Esta iniciativa no solo busca reforzar nuestras competencias en el manejo de tecnologías y herramientas de bases de datos relacionales, sino que representa un paso significativo hacia la consolidación de un entendimiento técnico más sofisticado y alineado con los estándares de la industria. La elección de este modelo en particular nos permite adentrarnos en las complejidades del diseño de esquemas de datos optimizados, la normalización de tablas para evitar redundancias, y la implementación de relaciones lógicas que garanticen la integridad y consistencia de la información, aspectos todos críticos en cualquier sistema de información comercial real.

Este proyecto busca elevar nuestra comprensión a un nivel marcadamente profesional, que no solo cumpla con las consignas académicas establecidas, sino que nos prepare para los desafíos y exigencias de los entornos laborales profesionales del futuro. En un panorama tecnológico donde el manejo eficiente de datos de productos es crucial para la logística, el control de inventario, la trazabilidad y la experiencia del cliente, dominar la arquitectura de bases de datos detrás de estos sistemas se convierte en una competencia invaluable. A través de este ejercicio, aspiramos a cerrar la brecha entre el conocimiento teórico y las demandas prácticas del sector, familiarizándonos con flujos de trabajo, mejores prácticas y criterios de optimización que son el pan de cada día en el desarrollo de software y la administración de sistemas a nivel corporativo, sentando así una base sólida para nuestra incursión y éxito en el campo profesional.

2. Etapa 1 – Modelado y Definición de Constraints:



El diagrama muestra una base de datos bastante simple, con dos tablas que están directamente relacionadas: **productos** y **código_barras**.

La idea es separar la información del producto de la información del código de barras. Por un lado, en la tabla **código_barras**, se guardan los datos técnicos del código: el número, el tipo, la fecha en que se asignó y algunas observaciones. También tiene un campo que indica si está activo o no.

En resumen, esta tabla se encarga de identificar de forma única a cada producto.

Por otro lado, la tabla **productos** guarda los datos más comunes del artículo: el nombre, la marca, la categoría, el precio y el peso. Además, también tiene un campo llamado **eliminado**, que sirve para marcar si el producto sigue disponible o fue dado de baja.

La parte importante está en cómo se conectan las dos tablas. Dentro de **productos** hay un campo llamado **codigo_barras_id**, que es una clave foránea. Ese campo apunta al **id** de la tabla **código_barras**, lo que significa que cada producto tiene su propio código asignado.

Y como ese campo es único, también se asegura que un mismo código de barras no se repita en otro producto.

En pocas palabras, la relación entre ambas tablas es de **uno a uno**:

un producto tiene un solo código de barras,

y un código de barras pertenece a un solo producto.

Este diseño nos es útil porque mantiene los datos más ordenados y facilita la gestión. Si más adelante se quiere modificar cómo se manejan los códigos de barras, no hace falta tocar la información de los productos en sí.

Creación del modelo relacional

La Etapa 1 se centra en la creación de un modelo relacional robusto para nuestra base de datos tpi_productos, aplicando todas las restricciones de integridad necesarias. Para lograrlo, se desarrollaron dos scripts SQL clave.

Creación del Esquema y Constraints (schemaTpiProductoCodigoDeBarras.sql)

Este script establece la estructura fundamental de la base de datos, definiendo las tablas codigos_barras (entidad B) y productos (entidad A) junto con sus respectivas restricciones, cumpliendo los requisitos:

- Definición de Tablas y Dominios: Se especifican las columnas con sus tipos de datos (BIGINT, BOOLEAN, ENUM, VARCHAR, DECIMAL, DATE), longitudes y restricciones de nulidad (NOT NULL), definiendo así el dominio válido para cada atributo. El uso de ENUM para codigos_barras.tipo es un claro ejemplo de restricción de dominio.
- Claves Primarias (PK): Ambas tablas cuentan con una columna id definida como BIGINT AUTO_INCREMENT PRIMARY KEY, garantizando un identificador único para cada registro.
- Restricción UNIQUE: Se aplica UNIQUE a codigos_barras.valor para asegurar que no existan dos códigos con el mismo número. Crucialmente, también se aplica UNIQUE a productos.codigos_barras_id para forzar la relación uno a uno (1:1) entre las tablas.
- Clave Foránea (FK): La columna productos.codigos_barras_id se define como FOREIGN KEY que referencia a codigos_barras(id), estableciendo formalmente la relación. Se utiliza ON DELETE SET NULL para mantener la integridad referencial si un código de barras es eliminado.
- Restricciones CHECK: Se implementaron múltiples reglas de negocio directamente en la base de datos:
 - En codigos_barras: CHECK (LENGTH(valor) > 0) evita valores vacíos.
 - En productos: CHECK (precio > 0), CHECK (peso >= 0), CHECK (nombre <> '') y CHECK (categoria IN (...)) aseguran la validez de los datos ingresados.
- Índices: Se crearon índices en columnas clave (eliminado, nombre, valor) para optimizar el rendimiento de futuras consultas, anticipando la Etapa 2.

Validación Práctica de Constraints (validacionConInserciones.sql)

Para cumplir con la actividad mínima de validación, este script demuestra la efectividad de las restricciones implementadas en el Script anterior:

- Inserciones Válidas: Se ejecutan dos pares de inserciones (codigos_barras seguido de productos) que cumplen todas las reglas definidas. Su éxito confirma que el esquema acepta datos correctos.
- Inserciones Erradas: Se intentan dos inserciones diseñadas específicamente para fallar y probar distintas restricciones:
 1. Violación de UNIQUE: Se intenta insertar un codigos_barras con un valor ya existente. El script documenta el error esperado (Duplicate entry...), validando la restricción UNIQUE.
 2. Violación de CHECK: Se intenta insertar un productos con un precio negativo. El script documenta el error esperado (CHECK constraint 'productos_chk_1' failed), validando la restricción CHECK sobre el precio.

[Aquí](#) la conversación con la ia

3. Etapa 2, Generación y carga de datos masivos con SQL puro

Descripción conceptual del mecanismo de generación de datos masivos

El documento describe un mecanismo para la generación masiva de datos en MySQL, utilizando un procedimiento almacenado que crea registros de forma iterativa con funciones pseudoaleatorias y estructuras de control internas. Este método evita el uso de herramientas externas, asegurando la integridad referencial, la coherencia y la reproducibilidad del proceso.

El procedimiento "GenerarDatosMasivos" crea una cantidad específica de registros utilizando un bucle WHILE. Dentro del bucle, se asignan valores aleatorios a los atributos del producto (nombre, marca, categoría, precio, peso y código de barras) mediante la función RAND(), lo que garantiza la variabilidad y el realismo de los datos.

Para la selección de valores categóricos (como marcas y categorías), el procedimiento utiliza listas internas con valores separados por comas. Mediante expresiones con SUBSTRING_INDEX y RAND(), se elige un elemento al azar, simulando tablas maestras sin crearlas físicamente. Esto simplifica la estructura del mecanismo y mantiene la diversidad de datos.

La integridad y las cardinalidades entre las tablas se garantizan mediante un orden de inserción lógico y el uso de claves generadas automáticamente. En cada iteración, se inserta primero un registro en la tabla codigos_barras, almacenando la información asociada al tipo, valor y fecha de asignación del código. Posteriormente, mediante la función LAST_INSERT_ID(), se obtiene el identificador recién creado y se utiliza como clave foránea al insertar el producto correspondiente en la tabla productos. De este modo, se asegura una relación uno a uno (1:1) entre cada producto y su código de barras, evitando inconsistencias referenciales.

Asimismo, el procedimiento implementa prácticas de optimización y control transaccional para mejorar el rendimiento. Antes de iniciar la carga masiva, se deshabilitan temporalmente las verificaciones de unicidad y claves foráneas, y se establecen bloques de confirmación (COMMIT) cada 1000 inserciones, lo que permite reducir la carga en memoria y mantener la estabilidad de la transacción. Estas medidas permiten generar más de medio millón de registros de manera eficiente, sin comprometer la integridad de la base.

En síntesis, este mecanismo combina la generación pseudoaleatoria con el control estructurado que ofrece SQL, logrando un equilibrio entre simplicidad, eficiencia y consistencia. La utilización de un procedimiento almacenado iterativo permite automatizar completamente el proceso de creación de datos masivos, garantizando que las tablas mantengan las relaciones definidas por el modelo lógico y que los datos resultantes puedan ser utilizados tanto para pruebas funcionales como de rendimiento.

[Aquí](#) la conversación con la ia.

The screenshot displays the SQL Server Enterprise Manager interface. The top pane shows a script named 'scriptParaGenerarDatosMasivos...' with the following SQL code:

```
76
77 -- Restaurar configuraciones
78 SET foreign_key_checks = 1;
79 SET unique_checks = 1;
80 SET autocommit = 1;
81
82 END$$
83
84 -- Restauramos el delimitador normal
85 DELIMITER ;
86
87 • CALL GenerarDatosMasivos(700000);
88
```

The bottom pane shows the 'Output' window with the 'Action Output' tab selected. It displays a table of execution results:

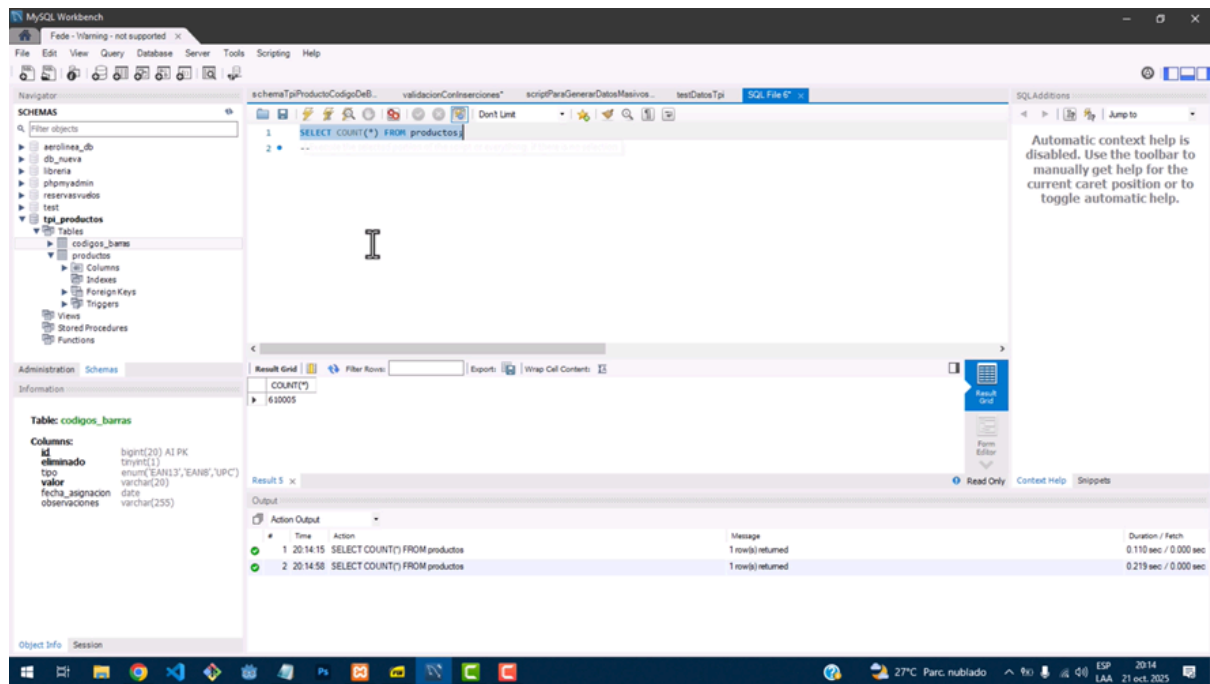
#	Time	Action	Message	Duration / Fetch
12	11:36:11	CREATE PROCEDURE GenerarDatosMasivos(IN cantidad INT) BEGIN DECLARE i INT ...	0 row(s) affected	0.000 sec
13	11:36:11	CALL GenerarDatosMasivos(700000)	1 row(s) returned	13.094 sec / 0.000 sec
14	11:36:37	CALL GenerarDatosMasivos(700000)	1 row(s) returned	- / 0.000 sec
15	11:36:50	CALL GenerarDatosMasivos(700000)	1 row(s) returned	- / 0.000 sec
16	11:37:03	CALL GenerarDatosMasivos(700000)	1 row(s) returned	- / 0.000 sec
17	11:37:17	CALL GenerarDatosMasivos(700000)	1 row(s) returned	- / 0.000 sec
18	11:37:30	CALL GenerarDatosMasivos(700000)	1 row(s) returned	- / 0.000 sec
19	11:37:44	CALL GenerarDatosMasivos(700000)	1 row(s) returned	- / 0.000 sec

Below the screenshot, there is a small progress bar and a status message:

progreso
Insertados 700000 registros

Análisis de Rendimiento: Impacto de los Índices en Consultas Masivas

Para cumplir con uno de los objetivos clave de la Etapa 2, realizamos una prueba específica para medir el impacto real de los índices en el rendimiento de las consultas sobre nuestra tabla productos, ahora poblada con más de 600,000 registros.



Nos enfocamos en el índice `idx_producto_nombre`, creado en la Etapa 1 sobre la columna `nombre`.

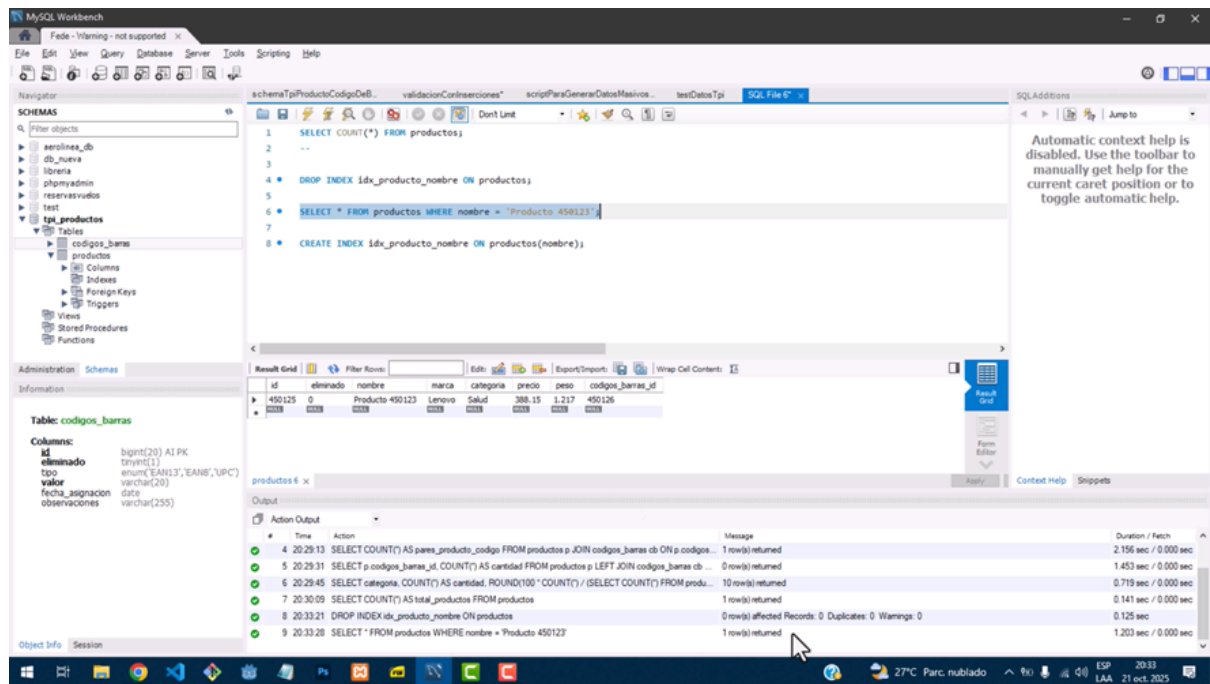
Metodología de Prueba:

La prueba consistió en ejecutar la misma consulta de búsqueda por nombre (`SELECT * FROM productos WHERE nombre = '...'`) en dos escenarios controlados:

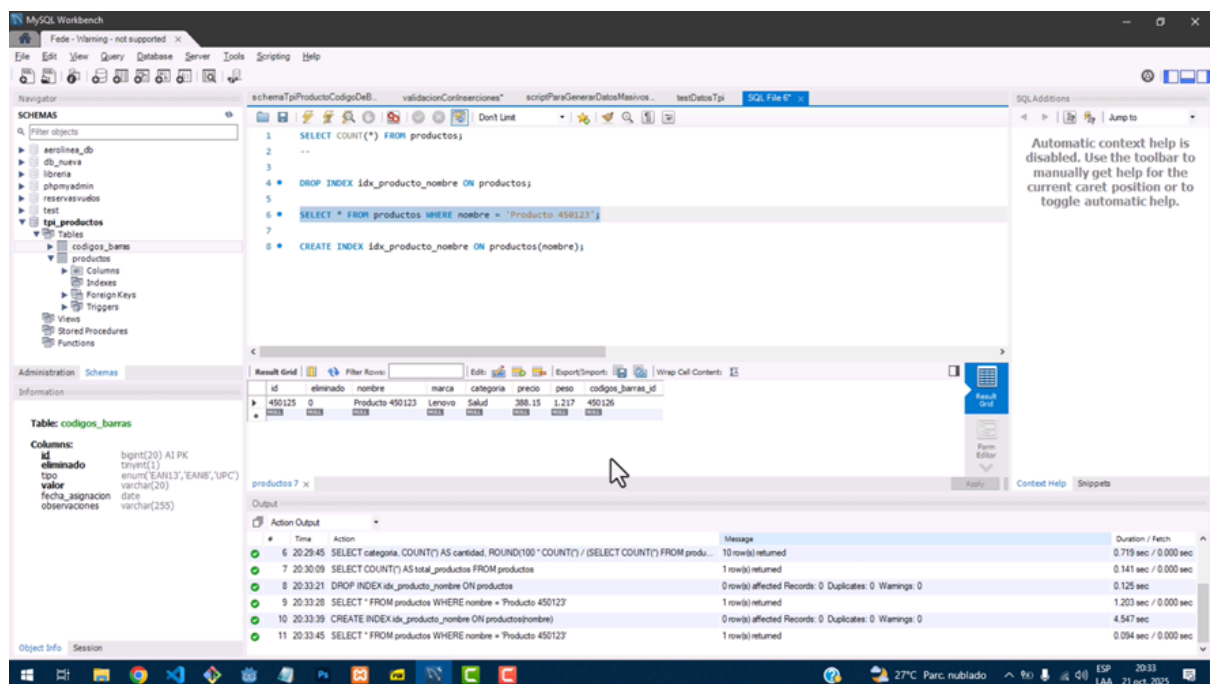
1. Escenario SIN Índice: Primero, eliminamos explícitamente el índice `idx_producto_nombre` de la tabla `productos` usando la sentencia `DROP INDEX idx_producto_nombre ON productos;`
2. Escenario CON Índice: Luego, volvimos a crear el índice usando `CREATE INDEX idx_producto_nombre ON productos(nombre);`

Ejecución y Resultados:

- Prueba 1 (Sin Índice): Tras eliminar el índice, ejecutamos la consulta `SELECT * FROM productos WHERE nombre = 'Producto 450123';`
 - Observación: Se evidenció que MySQL tuvo que realizar un **Full Table Scan**, examinando secuencialmente una gran cantidad de filas para encontrar el registro deseado.
 - Tiempo de Ejecución: El tiempo registrado fue de 1.203 segundos.



- **Prueba 2 (Con Índice):** Después de recrear el índice `idx_producto_nombre`, ejecutamos exactamente la misma consulta `SELECT` anterior.
 - Observación: Con el índice presente, MySQL pudo localizar el registro de manera mucho más directa, utilizando el índice para acceder rápidamente a las filas correspondientes sin necesidad de escanear toda la tabla.
 - Tiempo de Ejecución: El tiempo registrado fue significativamente menor: 0.094 segundos.



Conclusión del Análisis:

La diferencia en los tiempos de ejecución entre los dos escenarios es drástica. La presencia del índice `idx_producto_nombre` redujo el tiempo de respuesta de la consulta. Calculando el porcentaje de mejora:

- Reducción de tiempo: $1.203 - 0.094 = 1.109$ segundos
- Porcentaje de mejora: $\{1.109\} / \{1.203\} \times 100 \approx 92.2 \%$

El porcentaje de mejora es aproximadamente **92.2%**.

Este resultado demuestra empíricamente la importancia crítica de los índices en bases de datos con volúmenes significativos de información. Mientras que sin el índice la búsqueda requiere un escaneo completo y costoso de la tabla, con el índice la operación se vuelve prácticamente instantánea. Esto valida la decisión de diseño de incluir índices en la Etapa 1 y subraya su rol fundamental para asegurar la eficiencia y escalabilidad de la aplicación, tal como se busca en entornos profesionales.

4. Etapla 3: Consultas complejas y útiles a partir del CRUD inicial

Consulta 1:

Listado de productos con sus códigos de barras

Esta consulta muestra todos los productos junto con la información de su código de barras correspondiente. Utiliza un JOIN entre las tablas productos y codigos_barras para vincular cada producto con su código. Filtra los productos que no fueron eliminados y los ordena alfabéticamente por nombre.

	id_producto	nombre	marca	categoria	precio	tipo_codigo	valor_codigo	fecha_asignacion
►	194	Avanzado Adidas Alimentación	Zara	Deportes	152.02	UPC	840000000193	2025-06-10
	1945	Avanzado Adidas Alimentación	Nestle	Libros	840.61	EAN8	770000001944	2025-01-03
	2438	Avanzado Adidas Alimentación	Dell	Libros	194.90	UPC	7700000002437	2025-08-23
	3174	Avanzado Adidas Alimentación	Under Armour	Libros	783.41	EAN8	770000003173	2025-07-26
	4347	Avanzado Adidas Alimentación	Puma	Hogar	575.37	EAN8	770000004346	2024-11-04
	5222	Avanzado Adidas Alimentación	Under Armour	Libros	1050.32	UPC	840000005221	2025-04-05
	5344	Avanzado Adidas Alimentación	Under Armour	Alimentación	523.41	EAN8	840000005343	2024-11-16
	5348	Avanzado Adidas Alimentación	Under Armour	Alimentación	465.28	UPC	840000005347	2025-06-11
	6062	Avanzado Adidas Alimentación	Thom	Hogar	84.16	EAN8	840000006062	2025-08-22

Consulta 2:

Productos con precio mayor a 500 agrupados por categoría

Esta consulta obtiene las categorías de productos cuyo precio es superior a 500. Agrupa los resultados por categoría, contando cuántos productos caros hay en cada una y calculando el precio promedio. Finalmente, los ordena de mayor a menor promedio de precio.

	categoria	cantidad_productos_caros	promedio_precio
►	Salud	25745	1208.58
	Ropa	25994	1198.35
	Deportes	25963	1187.89
	Bebidas	25805	1163.31
	Hogar	25556	1141.54
	Libros	25728	1140.17
	Alimentación	25665	1113.68
	Muebles	25756	1093.80
	Electrónica	25522	1088.88

Consulta 3:

Marcas con presencia en más de tres categorías

La consulta identifica las marcas que fabrican productos en más de tres categorías diferentes. Utiliza GROUP BY y HAVING para agrupar por marca y aplicar la condición sobre

la cantidad de categorías. También muestra el total de productos y ordena por número de categorías y cantidad total.

	marca	total_productos	categorias_distintas
▶	Apple	46878	10
	Nestle	46852	10
	Samsung	46780	10
	Dell	46776	10
	Adidas	46772	10
	Ikea	46750	10
	LG	46748	10
	CocaCola	46715	10

Consulta 4:

Productos con precio superior al promedio general

Esta consulta selecciona los productos cuyo precio es mayor al promedio total de la tabla productos. Usa una subconsulta para calcular dicho promedio y luego compara cada precio con ese valor. Muestra los 50 productos más caros ordenados en forma descendente por precio.

	id	nombre	categoria	precio
▶	662816	Clásico Apple Libros	Bebidas	4999.95
	659016	Clásico Apple Bebidas	Bebidas	4999.89
	345013	Nuevo HP Hogar	Bebidas	4999.83
	600735	Premium Dell Deportes	Deportes	4999.81
	534286	Pro Under Armour Hogar	Deportes	4999.73
	435951	Eco Zara Libros	Juguetes	4999.66
	472652	Pro Apple Juguetes	Hogar	4999.63
	653246	Ultra Dell Juguetes	Ropa	4999.55
	384381	Avanzado Dell Electrónica	Ropa	4999.51

Medición 1 Con Índice

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	productos	NULL	ref	idx_producto_eliminado	idx_producto_eliminado	1	const	347753	100.00	Using index

Medición 2 Sin Índice

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	productos	NULL	ALL	NULL	NULL	NULL	NULL	695506	100.00	Using where

Conclusión:

El uso del índice idx_producto_eliminado mejora significativamente la eficiencia de la consulta, ya que evita recorrer toda la tabla.

Con índice, MySQL accede directamente a las filas que cumplen la condición (eliminado = FALSE) usando el índice, reduciendo a la mitad la cantidad de filas evaluadas.

En cambio, sin índice, el motor debe leer todas las filas (Full Table Scan)

[Aqui](#) conversación con la ia.

5. Etapa 4: Seguridad e Integridad

Usuario con privilegios mínimos

Este Usuario solo puede acceder a las vistas públicas mencionadas a continuación.

```
CREATE USER IF NOT EXISTS 'usuario_tpi'@'localhost' IDENTIFIED BY '12345678';
FLUSH PRIVILEGES;

-- 2. Dar permisos mínimos: SELECT sólo sobre las vistas públicas
GRANT SELECT ON tpi_productos.vista_productos_publico TO 'usuario_tpi'@'localhost';
GRANT SELECT ON tpi_productos.vista_codigos_publico TO 'usuario_tpi'@'localhost';

GRANT EXECUTE ON PROCEDURE tpi_productos.sp_get_producto_por_nombre TO 'usuario_tpi'@'localhost';

FLUSH PRIVILEGES;
```

Dos vistas para ocultar datos sensibles:

Productos Públicos

5 • `SELECT * FROM vista_productos_publico LIMIT 10;`

6

nombre	marca	categoria	precio	peso	tipo_codigo	codigo_valor
Ultra HP Alimentación	CocaCola	Juguetes	153.33	0.571	UPC	770000000000
Eco LG Ropa	Ikea	Muebles	76.07	4.028	EAN8	840000000001
Nuevo CocaCola Hogar	Under Armour	Deportes	65.31	0.046	UPC	770000000002
Portátil CocaCola Muebles	Zara	Alimentación	1184.59	3.292	UPC	840000000003
Nuevo Zara Muebles	Puma	Alimentación	128.94	0.002	EAN8	770000000004

Códigos de Barras Públicos

7 • `SELECT * FROM vista_codigos_publico LIMIT 10;`

valor	tipo	estado
770000000000	UPC	Código válido
840000000001	EAN8	Código válido
770000000002	UPC	Código válido
840000000003	UPC	Código válido
770000000004	EAN8	Código válido

Consulta segura

Esta consulta lo que hace es que con el "CONCAT" trata a la consulta como un dato y no como código sql, evitando así a los usuarios con malas intenciones y conocimiento de sql.

```
BEGIN
SELECT
    p.nombre,
    p.marca,
    p.categoria,
    p.precio,
    cb.valor AS codigo_barras
FROM productos p
JOIN codigos_barras cb ON p.codigos_barras_id = cb.id
WHERE p.nombre LIKE CONCAT('%', nombre_buscado, '%')
AND p.eliminado = FALSE;
END$$
```

9 • `CALL sp_get_producto_por_nombre('Sony');`

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	nombre	marca	categoria	precio	codigo_barras
▶	Pro Sony Deportes	Adidas	Hogar	50.28	840000000027
	Premium Sony Juguetes	Nestle	Bebidas	137.65	840000000053
	Nuevo Sony Alimentación	Dell	Deportes	1154.95	840000000054
	Smart Sony Juguetes	Dell	Bebidas	657.03	770000000076
	Smart Sony Electrónicos	Sony	Alimentación	182.61	840000000097

Entrada Maliciosa

Aquí podemos ver el caso contrario, sin la protección que nos da "CONCAT", en que un usuario malicioso podría destruir nuestra base de datos.

11 • `CALL sp_get_producto_por_nombre('' OR 1=1; DROP TABLE productos; --');`

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

nombre	marca	categoria	precio	codigo_barras
--------	-------	-----------	--------	---------------

Pruebas de integridad

Prueba 1: Se inserta un Código de Barras válido duplicado.

```
-- Se inserta un Código de Barras válido
INSERT INTO codigos_barras (tipo, valor, fecha_asignacion)
VALUES ('EAN13', '770000000007', CURDATE());
```

✖ 111 11:10:12 INSERT INTO codigos_barras (tipo, valor, fecha_asignacion) VALUES ('EAN13', '770000000000... Error Code: 1062. Duplicate entry '7700000000007' for key 'codigos_barras.valor'

Prueba 2: Se inserta un Producto válido duplicado.

```
-- Se inserta un Producto válido
INSERT INTO productos (nombre, marca, categoria, precio, peso, codigos_barras_id)
VALUES ('Auriculares BT', 'JBL', 'Electrónicos', 55.99, 0.350, 3);
```

✖ 121 11:10:24 INSERT INTO codigos_barras (tipo, valor, fecha_asignacion) VALUES ('EAN13', '770000000000... Error Code: 1062. Duplicate entry '7700000000007' for key 'codigos_barras.valor'

Prueba 3: Inserción con precio negativo.

```
-- Intento de inserción con precio negativo (-9.99)
INSERT INTO productos (nombre, marca, categoria, precio, peso, codigos_barras_id)
VALUES ('Bicicleta Fallida', 'Generico', 'Deportes', -9.99, 15.000, NULL);
```

✖ 127 11:11:59 INSERT INTO productos (nombre, marca, categoria, precio, peso, codigos_barras_id) VALUES (... Error Code: 3819. Check constraint 'productos_chk_1' is violated.

Prueba 4: Insertar un producto referenciando un codigos_barras_id que NO existe

```
-- Intentamos insertar un producto referenciando un codigos_barras_id que NO existe
INSERT INTO productos (nombre, marca, categoria, precio, peso, codigos_barras_id)
VALUES ('Producto con FK rota', 'Marca XYZ', 'Alimentación', 10.50, 0.500, 999999);
```

✖ 128 11:12:30 INSERT INTO productos (nombre, marca, categoria, precio, peso, codigos_barras_id) VALUES (... Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('tpi_productos'...

[Aqui](#) conversación con la ia.

6. Etapa 5. Concurrency y Transacciones.

En esta etapa de la actividad veremos como las bases de datos relacionales manejan situaciones en que dos o más sesiones intentan acceder a un mismo sector de una base de datos y como SQL y sus motores manejan la situación.

En primera instancia, antes de exponer las actividades requeridas y desarrolladas debemos definir en qué consiste la concurrencia y que son las transacciones:

Concurrencia: La concurrencia es la capacidad de ejecutar múltiples transacciones simultáneamente. Esto nos lleva necesariamente a saber que son las transacciones;

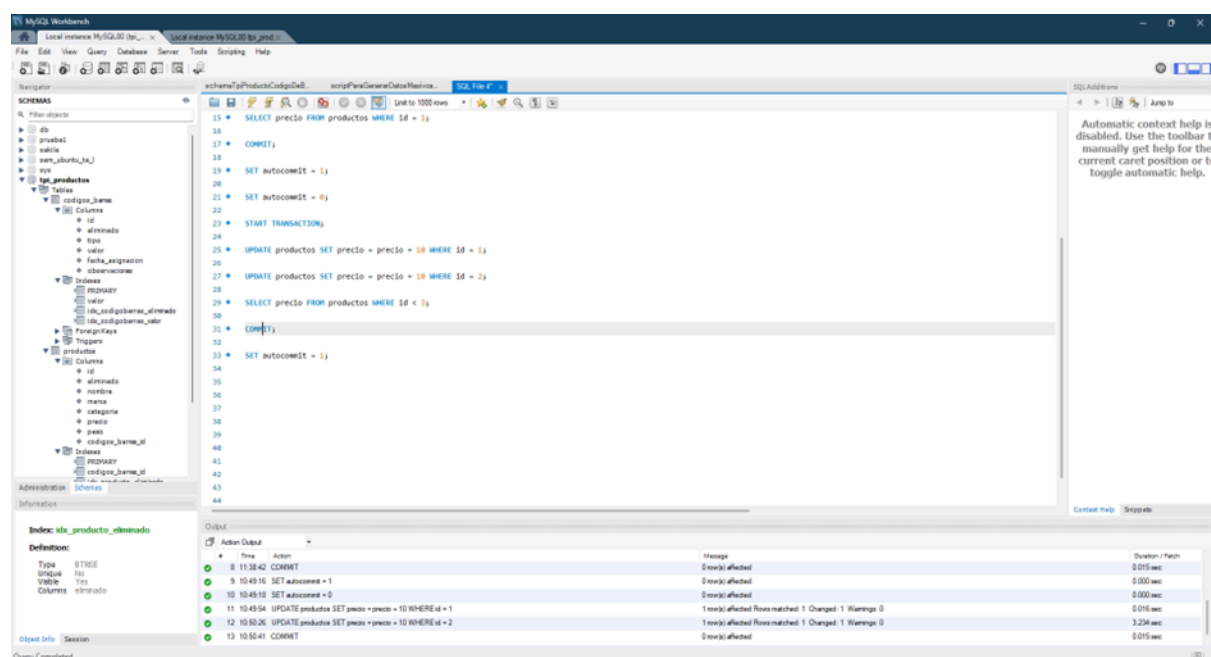
Transacciones: Una transacción en MySQL es una unidad lógica de trabajo que agrupa una o más operaciones SQL. Todas las operaciones de la transacción deben ejecutarse de forma completa (COMMIT) o no ejecutarse en absoluto (ROLLBACK), manteniendo la base de datos en un estado válido.

Con estas nociones, podemos empezar a desarrollar las actividades.

Simulación de deadlock:

Abrimos una sesión extra en MySQL Workbench y luego intentamos ingresar desde una a los productos con id 2 y 1, y simultáneamente, en la otra sesión, intentamos actualizar los productos 1 y 2.

Sesión A:



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'productos' selected. The main editor shows a SQL script with the following lines:

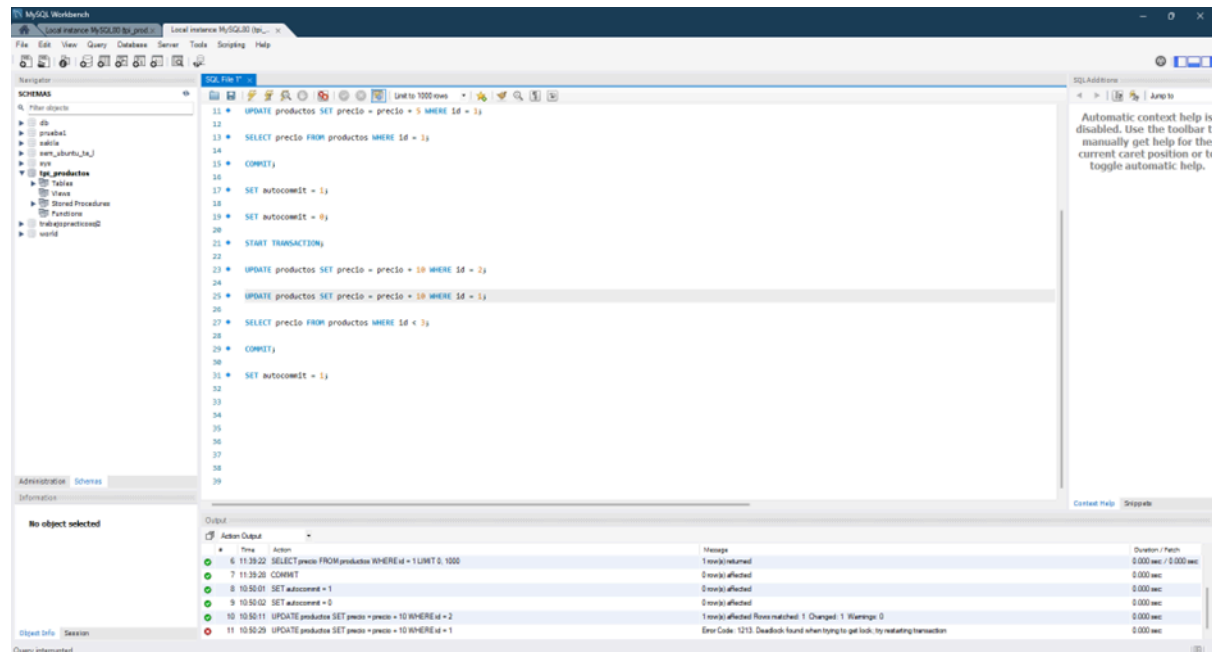
```
15 SELECT precio FROM productos WHERE id = 1;
16
17 COMMIT;
18
19 SET autocommit = 1;
20
21 SET autocommit = 0;
22
23 START TRANSACTION;
24
25 UPDATE productos SET precio = precio + 10 WHERE id = 1;
26
27 UPDATE productos SET precio = precio + 10 WHERE id = 2;
28
29 SELECT precio FROM productos WHERE id = 2;
30
31 COMMIT;
32
33 SET autocommit = 1;
34
35
36
37
38
39
40
41
42
43
44
```

The bottom panel shows the 'Output' tab with the following execution results:

Time	Action	Message	Duration / Path
8:11:38.42	COMMIT	0 rows(s) affected	0.015 sec
9:10:49:16	SET autocommit = 1	0 rows(s) affected	0.000 sec
10:10:49:18	SET autocommit = 0	0 rows(s) affected	0.000 sec
11:10:49:54	UPDATE productos SET precio = precio + 10 WHERE id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec
12:10:50:26	UPDATE productos SET precio = precio + 10 WHERE id = 2	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	3.234 sec
13:10:50:41	COMMIT	0 rows(s) affected	0.015 sec

Vemos que la transacción se completó correctamente, porque como veremos a continuación, el deadlock fue detectado y el motor decidió “matar” la sesión B.

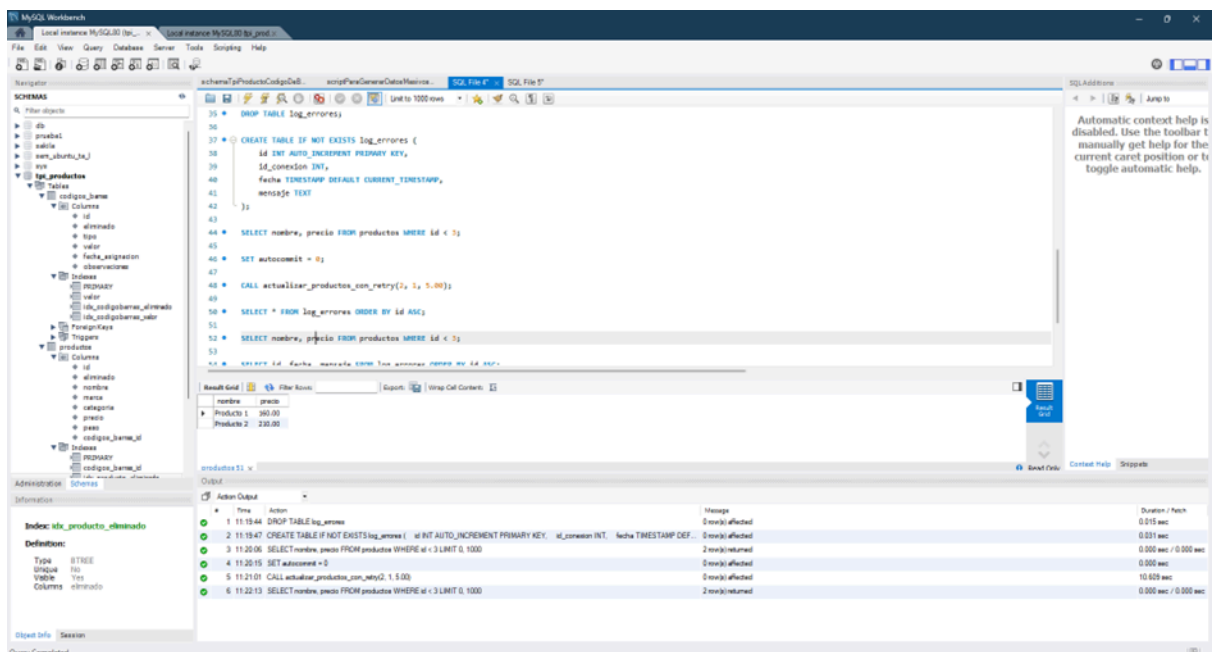
Sesión B:



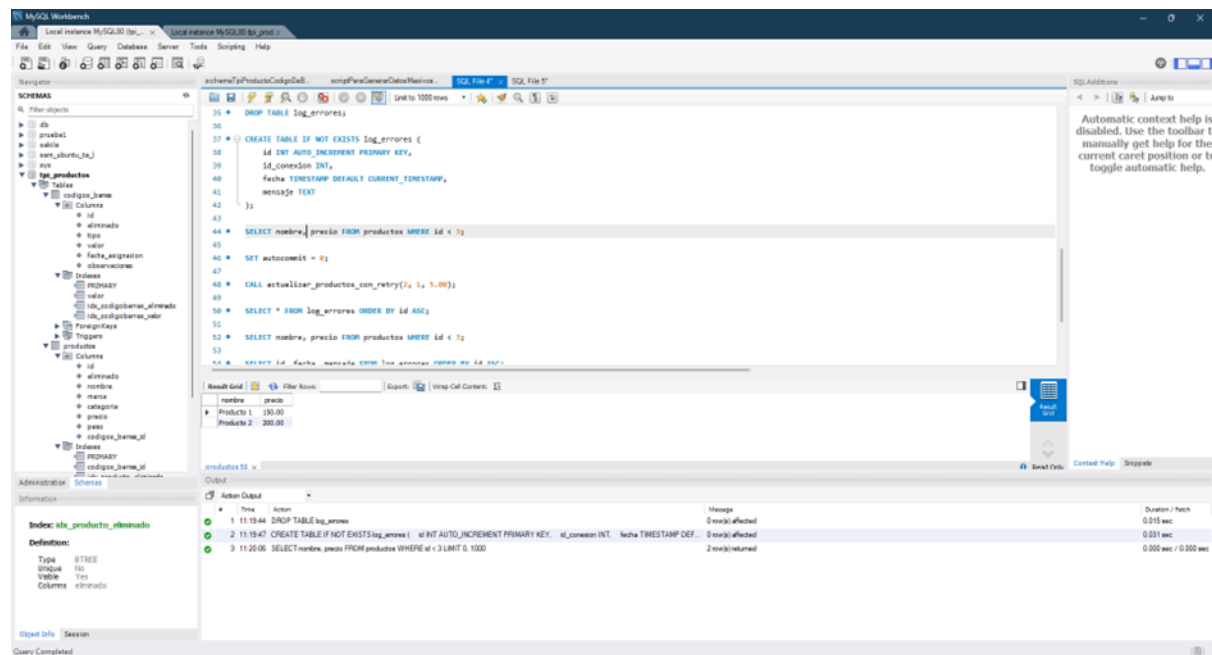
Implementación en SQL de transacciones completas:

Para este caso se implementó en SQL un procedimiento que maneja las transacciones completas, la creación de una tabla en la BD para registrar sus errores y movimientos, aplicando retry con backoff breve en el caso de deadlock.

Creación de una tabla para registrar movimientos:



Valores iniciales de los ítems a modificar:



The screenshot shows the MySQL Workbench interface with a SQL script being executed. The script includes the following queries:

```
35 DROP TABLE log_errores;
36
37 CREATE TABLE IF NOT EXISTS log_errores (
38   id INT AUTO_INCREMENT PRIMARY KEY,
39   id_conexion INT,
40   fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
41   mensaje TEXT
42 );
43
44 SELECT nombre, precio FROM productos WHERE id < 3;
45
46 SET autocommit = 0;
47
48 CALL actualizar_productos_con_retry(?, ?, 5.00);
49
50 SELECT * FROM log_errores ORDER BY id ASC;
51
52 SELECT nombre, precio FROM productos WHERE id < 3;
53
54 INSERT INTO log_errores (id_conexion, mensaje) VALUES (1, 'Error de conexión');
```

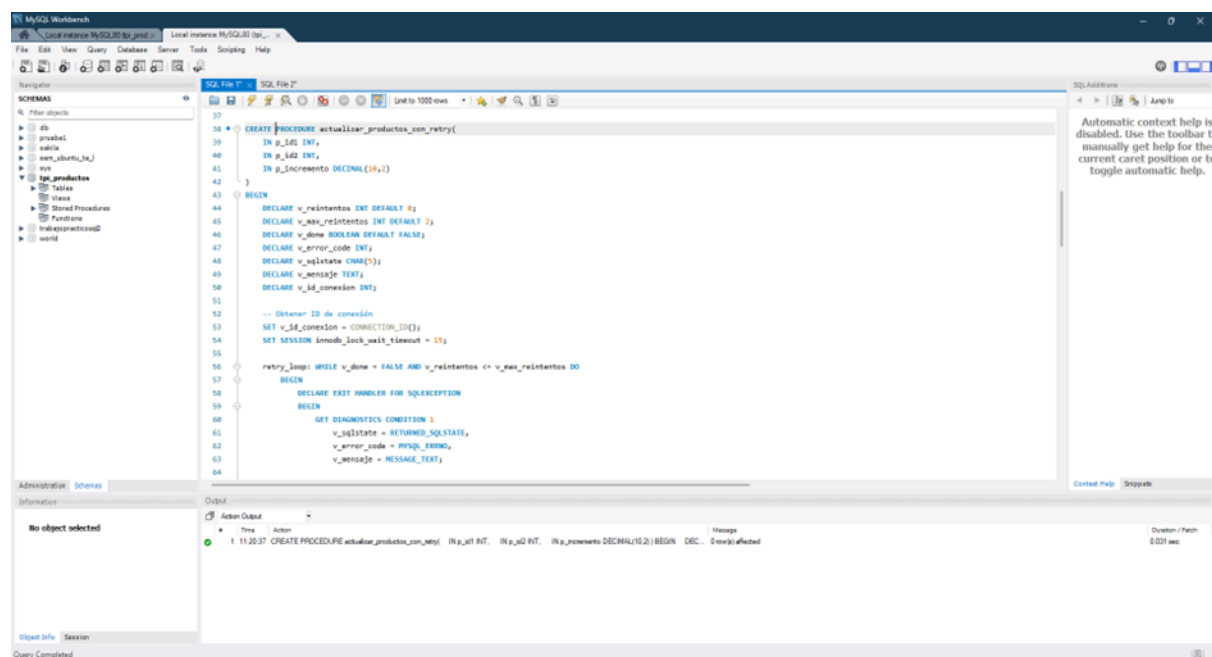
The results of the queries are displayed in the Output pane:

nombre	precio
Producto 1	100.00
Producto 2	200.00

The Action Output pane shows the execution details:

Time	Action	Message	Duration / Path
1 11:19:44	DROP TABLE log_errores	0 rows affected	0.015 sec
2 11:19:47	CREATE TABLE IF NOT EXISTS log_errores (id INT AUTO_INCREMENT PRIMARY KEY, id_conexion INT, fecha TIMESTAMP DEF...	0 rows affected	0.031 sec
3 11:20:06	SELECT nombre, precio FROM productos WHERE id < 3 LIMIT 0, 1000	2 rows returned	0.000 sec / 0.000 sec

Creación de procedimiento:



The screenshot shows the MySQL Workbench interface with a SQL script being executed. The script includes the following query:

```
37 CREATE PROCEDURE actualizar_productos_con_retry(
38   IN p_id INT,
39   IN p_incremento DECIMAL(10,2)
40 )
41 BEGIN
42
43   DECLARE v_reintentos INT DEFAULT 0;
44   DECLARE v_max_reintentos INT DEFAULT 3;
45   DECLARE v_errores BOOLEAN DEFAULT FALSE;
46   DECLARE v_estado CHAR(1);
47   DECLARE v_mensaje TEXT;
48   DECLARE v_id_conexion INT;
49
50   -- Obtener ID de conexión
51   SET v_id_conexion = CONNECTION_ID();
52   SET SESSION innodb_lock_wait_timeout = 15;
53
54   retry_loop: WHILE v_errores = TRUE AND v_reintentos <= v_max_reintentos DO
55     BEGIN
56       DECLARE EXIT HANDLER FOR SQLEXCEPTION
57       BEGIN
58         GET DIAGNOSTICS CONDITION 1
59           v_estado = RETURNED_SQLSTATE,
60           v_error_code = MYSQL_ERRNO,
61           v_mensaje = MESSAGE_TEXT;
62       END;
63     END;
64   END;
65
```

The Action Output pane shows the execution details:

Time	Action	Message	Duration / Path
1 11:20:37	CREATE PROCEDURE actualizar_productos_con_retry(IN p_id INT, IN p_incremento DECIMAL(10,2)) BEGIN DEC...	0 rows affected	0.031 sec

Llamado y ejecución del procedimiento en sesión A, argumentos 2, 1:

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

- Navigator:** Shows the database schema with tables like `tbl_producto` and `tbl_producto_eliminado`.
- Editor:** Contains the following SQL script:


```

      35 DROP TABLE log_errores;
      36
      37 CREATE TABLE IF NOT EXISTS log_errores (
      38   id INT AUTO_INCREMENT PRIMARY KEY,
      39   id_conexion INT,
      40   fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
      41   mensaje TEXT
      42 );
      43
      44 SELECT nombre, precio FROM productos WHERE id < 3;
      45
      46 SET autocommit = 0;
      47
      48 CALL actualizar_productos_con_retry(2, 1, 5.00);
      49
      50 SELECT * FROM log_errores ORDER BY id ASC;
      51
      52 SELECT nombre, precio FROM productos WHERE id < 3;
      53
      54 SELECT id, fecha, mensaje FROM log_errores ORDER BY id ASC;
      55
      56
      57
      58
      59
      60
      61
      
```
- Output:** Shows the execution results of the statements:

Time	Action	Message	Duration / Path
11:19:44	DROP TABLE log_errores	0 rows affected	0.015 sec
11:19:47	CREATE TABLE IF NOT EXISTS log_errores (id INT AUTO_INCREMENT PRIMARY KEY, id_conexion INT, fecha TIMESTAMP DEF...	0 rows affected	0.031 sec
11:20:06	SELECT nombre, precio FROM productos WHERE id < 3 LIMIT 0.1000	2 rows returned	0.000 sec / 0.000 sec
11:20:15	SET autocommit = 0	0 rows affected	0.000 sec
11:21:01	CALL actualizar_productos_con_retry(2, 1, 5.00)	0 rows affected	0.629 sec

Llamado y ejecución del procedimiento en sesión B, argumentos 1, 2:

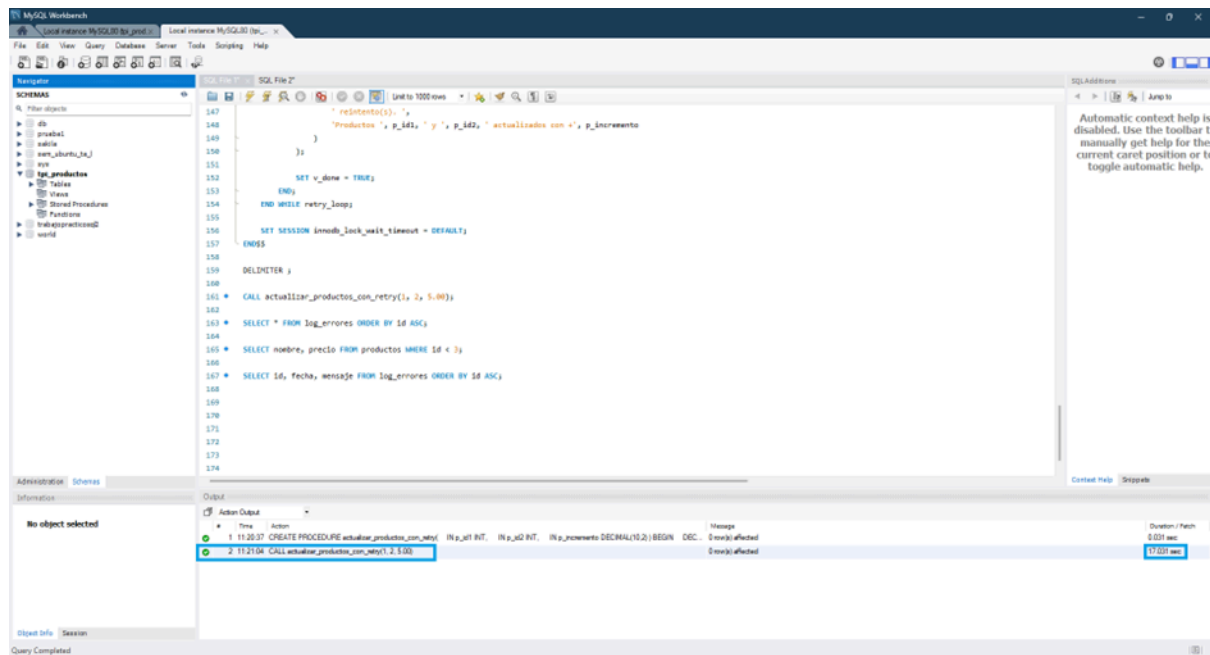
The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database schema with tables like `tbl_producto` and `tbl_producto_eliminado`.
- Editor:** Contains the following SQL script:

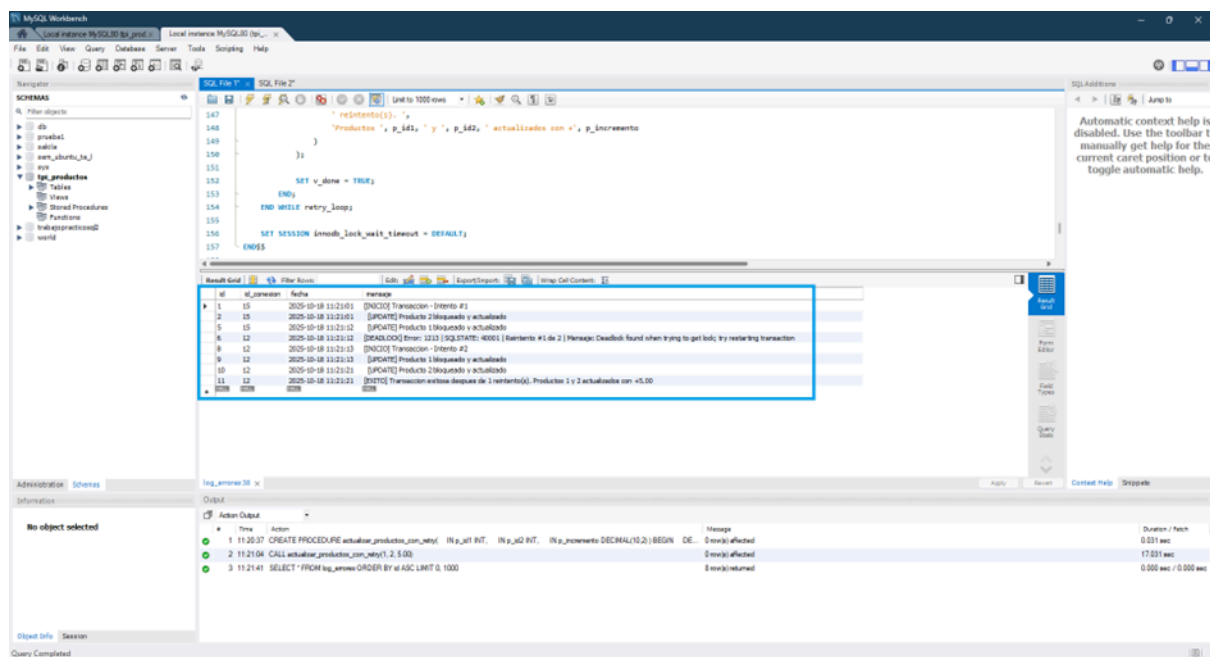

```

      147
      148
      149
      150
      151
      152
      153
      154
      155
      156
      157
      158
      159
      160
      161
      162
      163
      164
      165
      166
      167
      168
      169
      170
      171
      172
      173
      174
      
```
- Output:** Shows the execution results of the statements:

Time	Action	Message	Duration / Path
11:20:57	CREATE PROCEDURE actualizar_productos_con_retry(IN p_id INT, IN p_id INT, IN p_incremento DECIMAL(10,2)) BEGIN DEC...	0 rows affected	0.031 sec
11:21:04	CALL actualizar_productos_con_retry(1, 2, 0.00)	Running	1



Resultado de ejecución del proceso:



En ambas sesiones obtenemos las modificaciones realizadas efectivamente por las dos sesiones (5.00 por cada sesión) manejado para que puedan hacerse exitosamente por el procedimiento:

MySQL Workbench - Local instance MySQL80 (ip...)

Navigation: Filter objects, Schemas, Tables, Columns, Indexes, Triggers, Views, Procedures, Functions, Variables, Constants, World.

SQL Editor: schema7ProductsCódigoDel scriptParaGenerarDatosMasivos - SQL File 6 - SQL File 7

```

35 DROP TABLE log_errores;
36
37 CREATE TABLE IF NOT EXISTS log_errores (
38   id INT AUTO_INCREMENT PRIMARY KEY,
39   id_producto INT,
40   fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
41   mensaje TEXT
42 );
43
44 SELECT nombre, precio FROM productos WHERE id < 3;
45
46 SET autocommit = 0;
47
48 CALL actualizar_productos_con_retry(2, 1, 5.00);
49
50 SELECT * FROM log_errores ORDER BY id ASC;
51
52 SELECT nombre, precio FROM productos WHERE id < 3;
53
54 INSERT INTO log_errores (id_producto, fecha, mensaje) VALUES (2, NOW(), 'Producto 2 actualizado');

```

Result Grid:

nombre	precio
Producto 1	360.00
Producto 2	210.00

Output:

#	Time	Action	Message	Duration / Fech
1	11:19:44	DROP TABLE log_errores	0 rows affected	0.019 sec
2	11:19:47	CREATE TABLE IF NOT EXISTS log_errores (id INT AUTO_INCREMENT PRIMARY KEY, id_producto INT, fecha TIMESTAMP DEF	0 rows affected	0.031 sec
3	11:20:06	SELECT nombre, precio FROM productos WHERE id < 3 LIMIT 0, 1000	2 rows returned	0.000 sec / 0.000 sec
4	11:20:15	SET autocommit = 0	0 rows affected	0.000 sec
5	11:21:01	CALL actualizar_productos_con_retry(2, 1, 5.00)	0 rows affected	10.629 sec
6	11:22:13	SELECT nombre, precio FROM productos WHERE id < 3 LIMIT 0, 1000	2 rows returned	0.000 sec / 0.000 sec

Index: idx_producto_eliminado

Definition:

Type: BTREE

Unique: No

Visible: Yes

Columns: eliminado

Query Completed

MySQL Workbench - Local instance MySQL80 (ip...)

Navigation: Filter objects, Schemas, Tables, Columns, Indexes, Triggers, Views, Procedures, Functions, Variables, Constants, World.

SQL Editor: schema7ProductsCódigoDel scriptParaGenerarDatosMasivos - SQL File 6 - SQL File 7

```

149
150
151
152 SET v_done = TRUE;
153 END;
154 END WHILE retry_loop;
155
156 SET SESSION innodb_lock_wait_timeout = DEFAULT;
157
158
159 DELIMITER ;
160
161 CALL actualizar_productos_con_retry(1, 2, 5.00);
162
163 SELECT * FROM log_errores ORDER BY id ASC;
164
165 SELECT nombre, precio FROM productos WHERE id < 3;
166
167 SELECT id, fecha, mensaje FROM log_errores ORDER BY id ASC;
168
169

```

Result Grid:

nombre	precio
Producto 1	360.00
Producto 2	210.00

Output:

#	Time	Action	Message	Duration / Fech
1	11:20:37	CREATE PROCEDURE actualizar_productos_con_retry(IN p_id INT, IN p_id INT, IN p_incremento DECIMAL(10,2)) BEGIN DE	0 rows affected	0.031 sec
2	11:21:04	CALL actualizar_productos_con_retry(2, 5.00)	0 rows affected	17.031 sec
3	11:21:41	SELECT * FROM log_errores ORDER BY id ASC LIMIT 0, 1000	0 rows returned	0.000 sec / 0.000 sec
4	11:22:02	SELECT nombre, precio FROM productos WHERE id < 3 LIMIT 0, 1000	2 rows returned	0.000 sec / 0.000 sec

No object selected

Query Completed

Comparación de niveles de aislamiento.

Usaremos para este caso, dos sesiones nuevamente y compararemos a READ COMMITTED y REPEATABLE READ.

Tomamos un producto y fijamos su precio en 100.00.

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following queries:

```
1 USE tpj_productos;
2
3 UPDATE productos SET precio = 100.00 WHERE id = 1;
4
5 SELECT nombre, precio FROM productos WHERE id = 1;
6
7 -- Non-Repeatable Read (Read Committed)
8 SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
9
10 select @@transaction_isolation;
11
12 START TRANSACTION;
13
14 SELECT id, nombre, precio FROM productos WHERE id = 1;
15
16 COMMIT;
```

The Results grid shows the output of the SELECT query:

id	nombre	precio
1	Producto 1	100.00

The Output tab shows the execution log:

Time	Action	Message	Duration / Batch
1 10:38:43	UPDATE productos SET precio = 100.00 WHERE id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec
2 10:44:11	SELECT nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Seteamos en sesión A el nivel de aislamiento a READ COMMITTED.

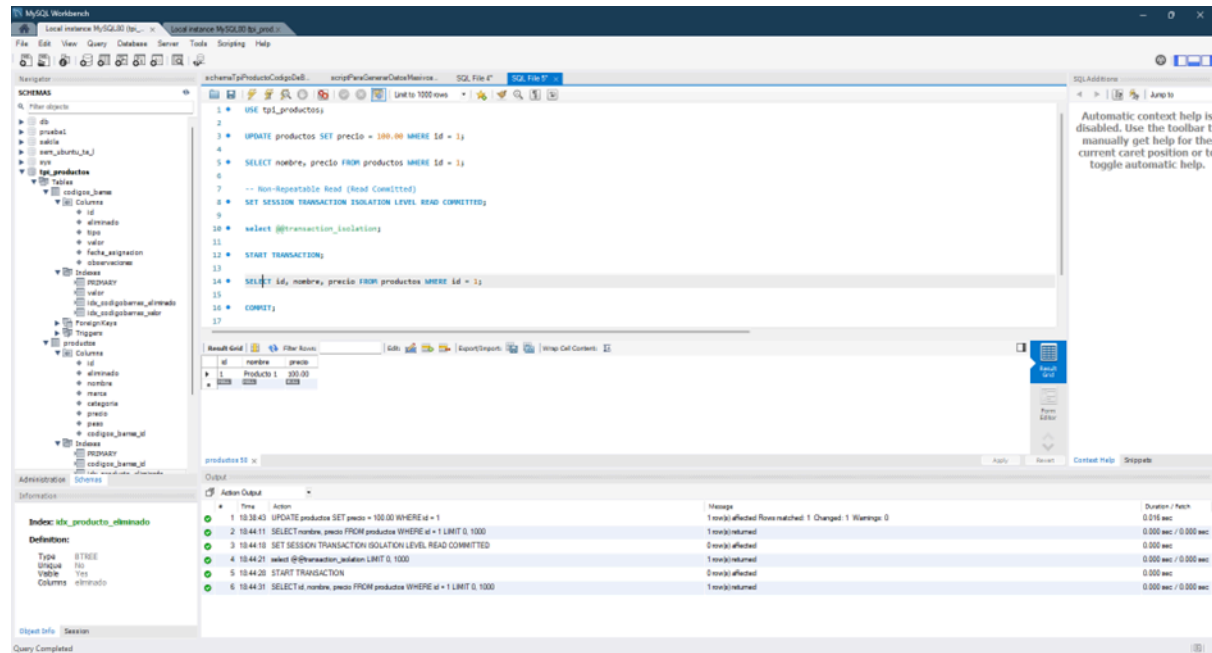
The screenshot shows the MySQL Workbench interface with the same SQL queries as the previous screenshot. The Results grid shows the output of the SELECT query:

id	nombre	precio
1	Producto 1	100.00

The Output tab shows the execution log:

Time	Action	Message	Duration / Batch
1 10:38:43	UPDATE productos SET precio = 100.00 WHERE id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec
2 10:44:11	SELECT nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
3 10:44:10	SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED	0 row(s) affected	0.000 sec
4 10:44:21	select @@transaction_isolation LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Comenzamos una transacción en sesión A, leemos precio 100.00:



The screenshot shows the MySQL Workbench interface. The left sidebar displays the database schema, including tables like `tbl_producto` and `tbl_producto_eliminado`. The central SQL editor contains the following code:

```
1 USE tp1_producto;
2
3 UPDATE productos SET precio = 100.00 WHERE id = 1;
4
5 SELECT nombre, precio FROM productos WHERE id = 1;
6
7 -- Non-Repeatable Read (Read Committed)
8 SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
9
10 select @@transaction_isolation;
11
12 START TRANSACTION;
13
14 SELECT id, nombre, precio FROM productos WHERE id = 1;
15
16 COMMIT;
```

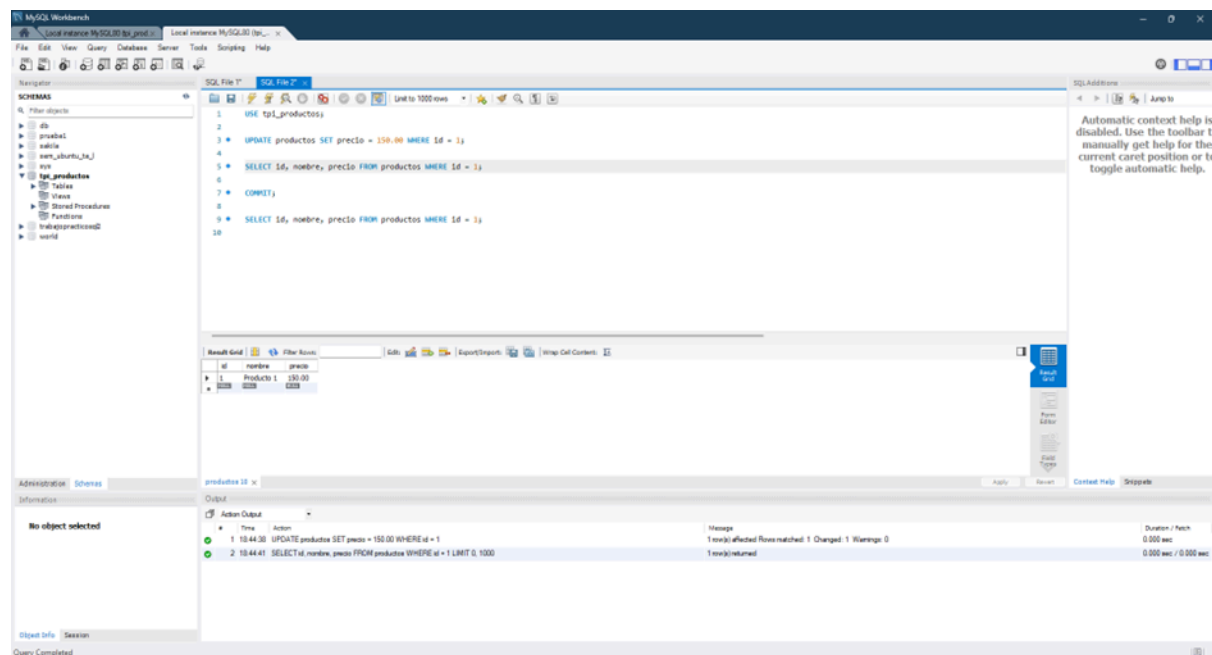
The 'Result Grid' shows the result of the `SELECT` query:

id	nombre	precio
1	Producto 1	100.00

The 'Output' pane shows the execution of the transaction steps:

Time	Action	Message	Duration / Fetch
10:44:43	UPDATE productos SET precio = 100.00 WHERE id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec
10:44:41	SELECT nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
10:44:40	SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED	0 row(s) affected	0.000 sec
10:44:41	select @@transaction_isolation LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
10:44:40	START TRANSACTION	0 row(s) affected	0.000 sec
10:44:41	SELECT id, nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

En sesión B cambiamos el precio a 150.00:



The screenshot shows the MySQL Workbench interface. The left sidebar displays the database schema. The central SQL editor contains the following code:

```
1 USE tp1_producto;
2
3 UPDATE productos SET precio = 150.00 WHERE id = 1;
4
5 SELECT id, nombre, precio FROM productos WHERE id = 1;
6
7 COMMIT;
8
9 SELECT id, nombre, precio FROM productos WHERE id = 1;
```

The 'Result Grid' shows the result of the `SELECT` query:

id	nombre	precio
1	Producto 1	150.00

The 'Output' pane shows the execution of the transaction steps:

Time	Action	Message	Duration / Fetch
10:44:40	UPDATE productos SET precio = 150.00 WHERE id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.000 sec
10:44:41	SELECT id, nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Volvemos a sesión A, leemos el precio nuevamente y como no hemos hecho commit en sesión B, seguimos leyendo precio 100.00:

The screenshot shows the MySQL Workbench interface. The left sidebar displays the database schema, including tables like `tbl_producto` and `tbl_producto_eliminado`. The main editor window shows the following SQL code:

```

1 USE tp1_productos;
2
3 UPDATE productos SET precio = 100.00 WHERE id = 1;
4
5 SELECT nombre, precio FROM productos WHERE id = 1;
6
7 -- Non-Repeatable Read (Read Committed)
8 SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
9
10 select @@transaction_isolation;
11
12 START TRANSACTION;
13
14 SELECT id, nombre, precio FROM productos WHERE id = 1;
15
16 COMMIT;
17

```

The output window shows the execution of the transaction. The first two queries (UPDATE and SELECT) are executed successfully. The third query (SELECT @@transaction_isolation) returns the value `READ COMMITTED`. The fourth query (START TRANSACTION) starts the transaction. The fifth query (SELECT id, nombre, precio FROM productos WHERE id = 1) returns the price of 100.00. The sixth query (COMMIT) commits the transaction.

Se hizo commit en sesión B, ahora sin salir de la transacción en A, obtenemos precio 150.00:

The screenshot shows the MySQL Workbench interface. The left sidebar displays the database schema, including tables like `tbl_producto` and `tbl_producto_eliminado`. The main editor window shows the same SQL code as the previous screenshot. The output window shows the execution of the transaction. The first two queries (UPDATE and SELECT) are executed successfully. The third query (SELECT @@transaction_isolation) returns the value `READ COMMITTED`. The fourth query (START TRANSACTION) starts the transaction. The fifth query (SELECT id, nombre, precio FROM productos WHERE id = 1) returns the price of 150.00. The sixth query (COMMIT) commits the transaction.

Cerramos la transacción en A, con un commit, quedando el valor en 150.00:

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following SQL statements:

```
3 * UPDATE productos SET precio = 100.00 WHERE id = 1;
4
5 * SELECT nombre, precio FROM productos WHERE id = 1;
6
7 -- Non-Repeatable Read (Read Committed)
8 * SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
9
10 select @@transaction_isolation;
11
12 START TRANSACTION;
13
14 * SELECT id, nombre, precio FROM productos WHERE id = 1;
15
16 COMMIT;
17
18 * SELECT id, nombre, precio FROM productos WHERE id = 1;
```

The result grid shows the output of the SELECT statement:

id	nombre	precio
1	Producto 1	150.00

The output pane shows the execution of the transaction:

Time	Action	Message	Duration / Patch
2 10:44:11	SELECT nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
3 10:44:18	SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED	0 row(s) affected	0.000 sec
4 10:44:21	select @@transaction_isolation LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
5 10:44:20	START TRANSACTION	0 row(s) affected	0.000 sec
6 10:44:31	SELECT id, nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
7 10:44:52	SELECT id, nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
8 10:45:04	SELECT id, nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
9 10:45:10	COMMIT	0 row(s) affected	0.000 sec
10 10:45:15	SELECT id, nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Repetiremos la operación, pero ahora con un nivel distinto de aislamiento.

Volvemos a setear el precio en 100.00 en sesión A:

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following SQL statements:

```
14 * SELECT id, nombre, precio FROM productos WHERE id = 1;
15
16 COMMIT;
17
18 * SELECT id, nombre, precio FROM productos WHERE id = 1;
19
20 -- Repeatable Read
21
22 * UPDATE productos SET precio = 100.00 WHERE id = 1;
23
24 COMMIT;
25
26 * SELECT nombre, precio FROM productos WHERE id = 1;
27
28 -- Repeatable Read
29 * SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
30
31 select @@transaction_isolation;
32 * START TRANSACTION;
```

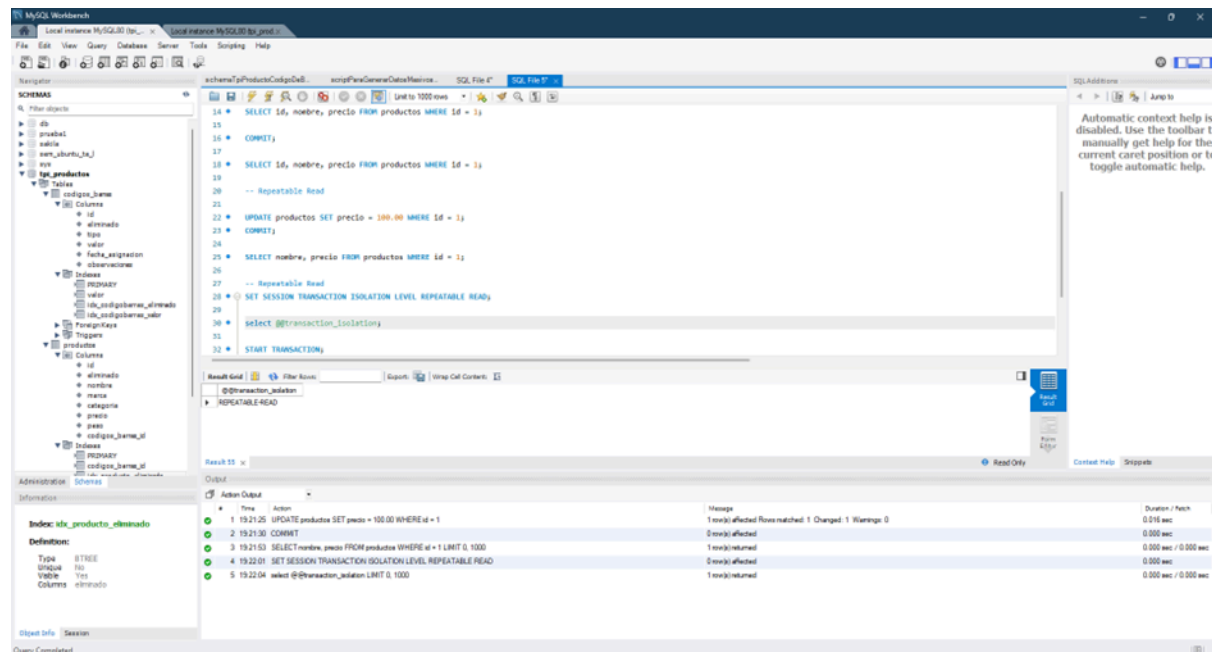
The result grid shows the output of the SELECT statement:

nombre	precio
Producto 1	100.00

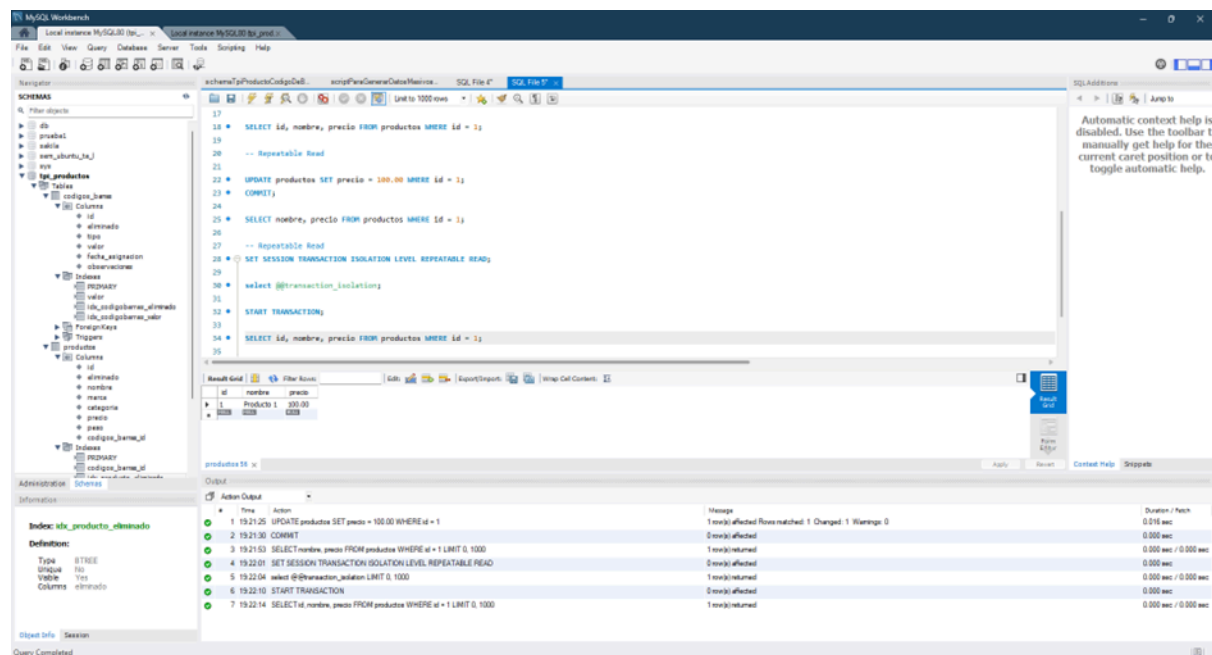
The output pane shows the execution of the transaction:

Time	Action	Message	Duration / Patch
1 10:21:25	UPDATE productos SET precio = 100.00 WHERE id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec
2 10:21:30	COMMIT	0 row(s) affected	0.000 sec
3 10:21:53	SELECT nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

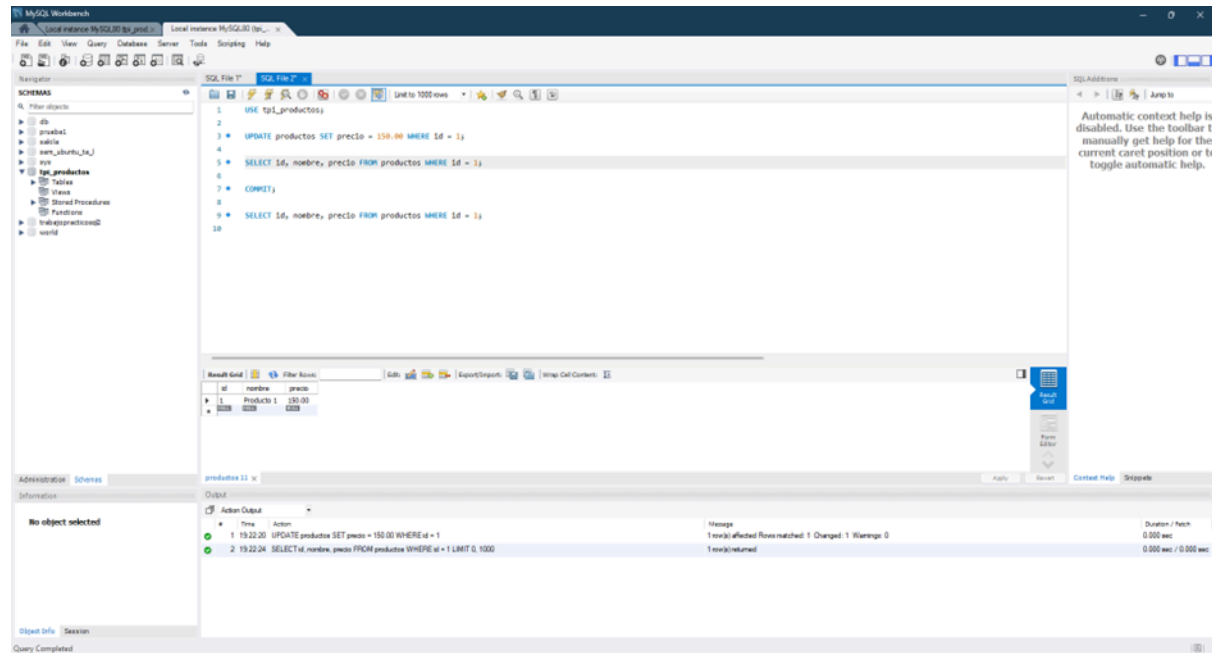
Establecemos el nivel de aislamiento en REPEATABLE READ:



Iniciamos una transacción en A, leyendo el producto con valor 100.00:



En sesión B modificamos el precio a 150.00, sin hacer commit:



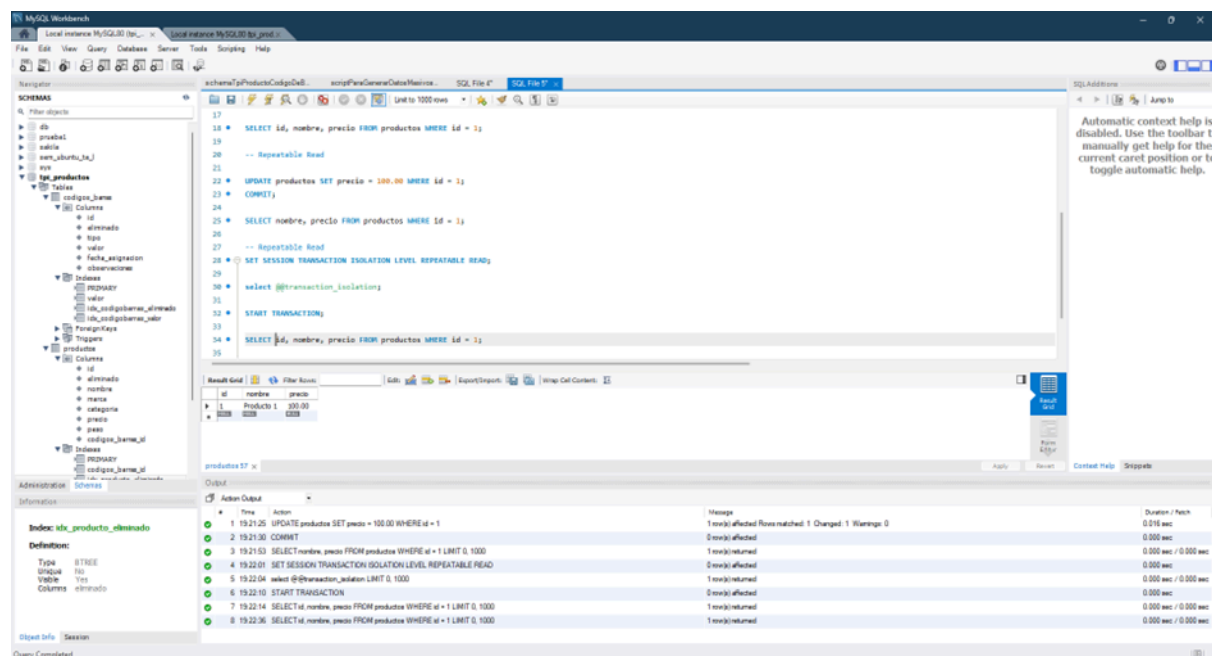
The screenshot shows the MySQL Workbench interface. The SQL editor contains the following script:

```
1 USE tpj_productos;
2
3 UPDATE productos SET precio = 150.00 WHERE id = 1;
4
5 SELECT id, nombre, precio FROM productos WHERE id = 1;
6
7 COMMIT;
8
9 SELECT id, nombre, precio FROM productos WHERE id = 1;
10
```

The output window shows the execution results:

#	Time	Action	Message	Duration / Patch
1	19:22:20	UPDATE productos SET precio = 150.00 WHERE id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.000 sec
2	19:22:24	SELECT id, nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Volvemos a A y como no se ha hecho commit de los cambios en B, leemos los mismos valores que en su inicio, 100.00, a pesar que se ha modificado la fila en B:



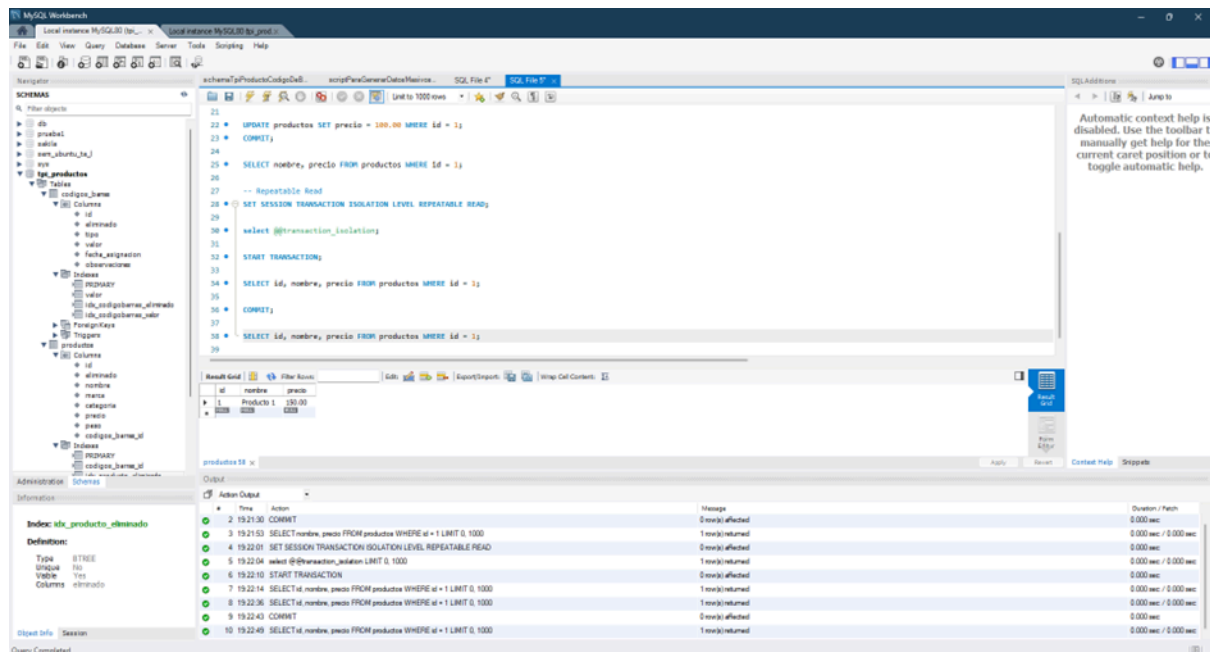
The screenshot shows the MySQL Workbench interface. The SQL editor contains the following script:

```
17
18 SELECT id, nombre, precio FROM productos WHERE id = 1;
19
20 -- Repeatable Read
21
22 UPDATE productos SET precio = 100.00 WHERE id = 1;
23
24 COMMIT;
25
26 SELECT nombre, precio FROM productos WHERE id = 1;
27
28 -- Repeatable Read
29
30 SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
31
32 select @@transaction_isolation;
33
34 START TRANSACTION;
35
36 SELECT id, nombre, precio FROM productos WHERE id = 1;
37
```

The output window shows the execution results:

#	Time	Action	Message	Duration / Patch
1	19:21:25	UPDATE productos SET precio = 100.00 WHERE id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec
2	19:21:30	COMMIT	0 row(s) affected	0.000 sec
3	19:21:53	SELECT nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
4	19:22:01	SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ	0 row(s) affected	0.000 sec
5	19:22:04	select @@transaction_isolation LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
6	19:22:10	START TRANSACTION	0 row(s) affected	0.000 sec
7	19:22:14	SELECT id, nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
8	19:22:36	SELECT id, nombre, precio FROM productos WHERE id = 1 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Hacemos commit en B y en A, y ahora si en A podremos ver el valor de 150.00:



Informe de concurrencia y transacciones.

A lo largo de esta sección vimos cómo en la práctica las bases de datos manejan los accesos concurrentes, lo cual aquí se simuló generando dos conexiones en MySQL Workbench. En primer lugar, vimos cómo se asegura siempre un estado válido de los datos. Cuando se quieren modificar las mismas filas desde dos fuentes distintas, el motor InnoDB resuelve el deadlock eligiendo una de ellas para completar su transacción, mientras que la otra es rechazada. Se lanza un error que nos permite manejarlo, volviendo al usuario a su estado anterior (rollback) y manteniendo así la consistencia en los datos almacenados en la DB.

Conociendo estos conceptos, podemos desarrollar soluciones que permitan a múltiples conexiones realizar sus modificaciones. Para ello existen los procedimientos almacenados, que pueden contener un flujo de trabajo para manejar errores, como en este caso, volver a intentar la transacción.

Finalmente, debemos tener en cuenta que el cambio de estado de la base de datos puede afectar las tareas de otras fuentes. Por ello, es crucial analizar los distintos niveles de aislamiento, como READ COMMITTED y REPEATABLE READ, para elegir el más adecuado y garantizar que los resultados de las consultas sean consistentes y seguros para los usuarios.

Interacción con IA.

A través de [este link](#), se encuentra accesible la interacción con el modelo Gemini 2.5 Pro para el desarrollo de esta sección.

7. Conclusión

En conclusión, el desarrollo de este proyecto nos ha permitido integrar de manera efectiva los conocimientos teóricos adquiridos con la aplicación práctica de herramientas especializadas, como MySQL Workbench, en la construcción de un modelo de base de datos orientado a la gestión de códigos de barras de productos. A través de este proceso, hemos reforzado nuestra comprensión sobre el diseño lógico y físico de bases de datos, la importancia de la normalización para evitar redundancias, y la implementación de relaciones que aseguren la integridad y consistencia de la información. Este aprendizaje no solo nos ha permitido comprender la relevancia de una arquitectura de datos bien estructurada, sino también apreciar su impacto directo en la eficiencia y confiabilidad de los sistemas de información comerciales.

Asimismo, este trabajo representa un avance significativo en nuestra formación profesional, al acercarnos a las prácticas y estándares utilizados en entornos laborales reales. El dominio de herramientas de gestión de bases de datos relacionales se revela como una competencia esencial en un mundo donde los datos son el núcleo de las operaciones empresariales. Gracias a este proyecto, hemos fortalecido nuestra capacidad para analizar, diseñar y optimizar soluciones tecnológicas alineadas con las necesidades del sector, sentando las bases para un desempeño sólido y competitivo en el ámbito de la ingeniería de software y la administración de sistemas.

8. Anexo

Utilización IA Parte 1: <https://chatgpt.com/share/68f647f7-4240-8001-ab78-041c5570c886>

Utilización IA Parte 2: <https://chatgpt.com/share/68f6430a-50d0-8001-a7e5-c883271763f5>

Utilización IA Parte 3: <https://chatgpt.com/share/68f8cda2-3218-8001-8266-4f9a54178017>

Utilización IA Parte 4: <https://chatgpt.com/share/68f8d6ef-b848-8001-9b84-67ff9e02ae10>

Utilización IA Parte 5: <https://gemini.google.com/share/c17439f2d90d>

Video explicativo entregable: https://youtu.be/JCD1Q-m4_KM?si=yaOb2_iOfDZiR0Zx