



TRABAJO FINAL INTEGRADOR – BASES DE DATOS I

INTEGRANTES: TADEO OSCAR ACOSTA, DANIEL ALFREDO OSCAR
ALDERETE, NAZARENO ARANDA, JULIAN BLANCO CORTÉS.

Resumen Ejecutivo

El presente Trabajo Final Integrador de la asignatura **Bases de Datos I** tiene como propósito aplicar de manera práctica e integrada los conceptos fundamentales del diseño, implementación, manipulación y administración de bases de datos relacionales.

A lo largo de las distintas etapas se abordan, de forma progresiva, los principales aspectos del ciclo de vida de una base de datos: modelado conceptual, normalización, definición de restricciones, generación de datos masivos, consultas, índices, seguridad y concurrencia.

En la **Etapa 1**, se diseñó el **modelo entidad–relación** correspondiente al dominio *Vehículo–SeguroVehicular*, estableciendo las reglas de negocio, cardinalidades y constraints necesarios para garantizar la integridad de los datos.

A partir de este modelo se construyó el esquema físico en MySQL, empleando claves primarias, foráneas, restricciones UNIQUE y CHECK, además de asegurar la idempotencia en los scripts mediante el uso de DROP IF EXISTS.

La **Etapa 2** se centró en la **generación y carga masiva de datos** utilizando exclusivamente SQL puro.

Se aplicaron técnicas de producto cartesiano, funciones aleatorias (RAND(), CONCAT(), LPAD()) y control de integridad referencial para poblar las tablas principales con **10 000 registros** (5 000 por entidad).

Este conjunto de datos sirvió como base para las pruebas de rendimiento y consistencia realizadas en etapas posteriores.

En la **Etapa 3**, se desarrollaron **consultas SQL complejas y vistas**, analizando el rendimiento de distintas variantes y evaluando el impacto del uso de **índices** sobre los tiempos de ejecución.

Se implementaron índices simples y compuestos, comparando las ejecuciones *con* y *sin índice*, y documentando las mejoras de performance mediante el comando EXPLAIN.

La **Etapa 4** abordó la **seguridad y gestión de usuarios**, estableciendo distintos niveles de acceso mediante la creación de roles y privilegios.

Se aplicaron principios de mínimo privilegio, controlando operaciones de lectura, inserción, actualización y eliminación según el perfil del usuario.

Además, se verificó la correcta aplicación de los permisos mediante pruebas de acceso con cuentas diferenciadas.

Finalmente, la **Etapa 5** se enfocó en la **concurrencia y manejo de transacciones**, simulando escenarios de acceso simultáneo y contención de recursos.

Se analizaron los efectos de distintos niveles de aislamiento (READ COMMITTED, REPEATABLE READ, SERIALIZABLE) y se validó la atomicidad y consistencia del sistema a través de operaciones START TRANSACTION, COMMIT y ROLLBACK.

Como resultado final, el trabajo integra un sistema relacional completo, idóneo para el dominio *Vehículo–SeguroVehicular*, construido sobre bases sólidas de diseño, rendimiento y seguridad.

El proceso incluyó además el uso pedagógico de herramientas de **Inteligencia Artificial como tutoría**, que asistieron en la exploración de estrategias, explicación conceptual y validación de resultados, sin intervenir en la producción literal de los scripts.

Este enfoque permitió consolidar competencias técnicas, analíticas y metodológicas en el uso de SQL, promoviendo una comprensión integral del funcionamiento y gestión de bases de datos reales.

ETAPA 1 – MODELO, ESQUEMA Y CONSTRAINTS

□ 1. Reglas de Negocio

1. Cada vehículo tiene **un único seguro vehicular** asociado.
2. Un seguro vehicular solo puede cubrir **un vehículo a la vez** (relación 1:1).
3. No puede existir un vehículo sin dominio, marca o modelo registrados.
4. No puede existir un seguro sin aseguradora, número de póliza ni tipo de cobertura.
5. El número de póliza, el dominio y el número de chasis deben ser **únicos**.
6. Las bajas se gestionan de forma **lógica**, mediante el campo booleano eliminado.
7. Las fechas de vencimiento de los seguros deben ser **posteriores a la fecha actual**.

□ 2. Modelo Entidad-Relación (DER)

El sistema se basa en dos entidades principales:

- **Vehiculo**
- **SeguroVehicular**

Relación: 1 a 1 (un vehículo posee un solo seguro vehicular).

Atributos principales:

- Vehículo: id, dominio, marca, modelo, año, nroChasis, eliminado, seguro_id (FK).

- SeguroVehicular: id, aseguradora, nroPoliza, cobertura, vencimiento, eliminado.

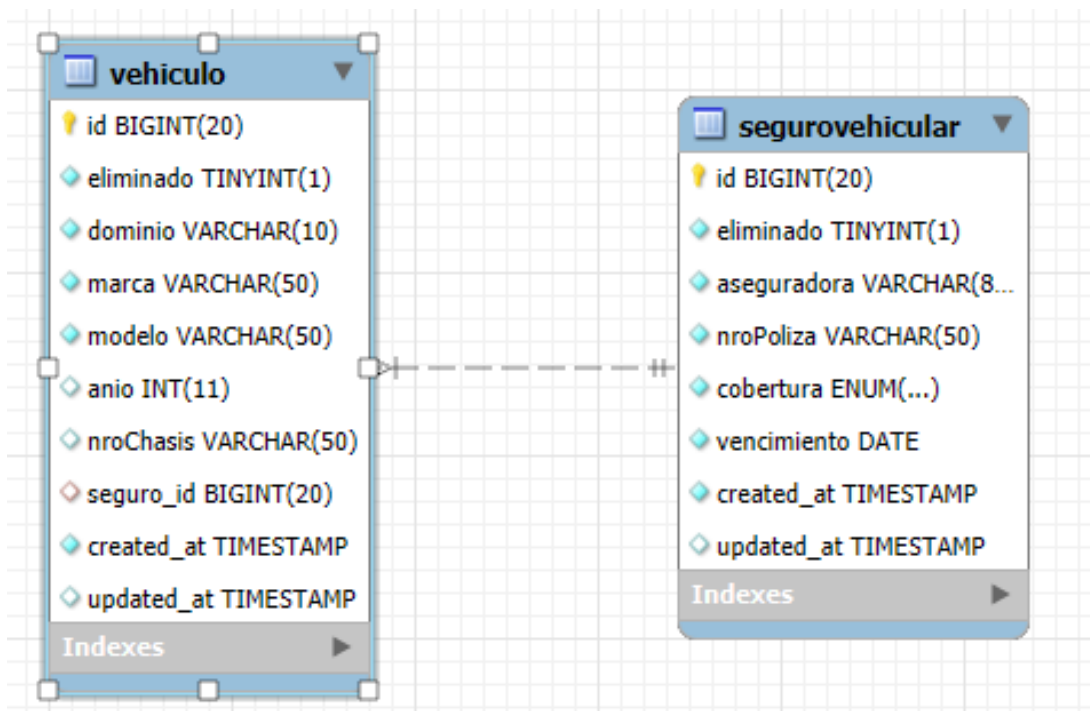


Figura 1. Diagrama Entidad–Relación entre Vehículo y SeguroVehicular.

⚙ 3. Modelo Relacional

Tabla	Campo	Tipo	Restricción
SeguroVehicular	id	BIGINT	PK, AUTO_INCREMENT
	aseguradora	VARCHAR(80)	NOT NULL
	nroPoliza	VARCHAR(50)	UNIQUE, NOT NULL
	cobertura	ENUM('RC', 'TERCEROS', 'TODO_RIESGO')	NOT NULL
	vencimiento	DATE	NOT NULL
	eliminado	BOOLEAN	DEFAULT FALSE
Vehiculo	id	BIGINT	PK, AUTO_INCREMENT
	dominio	VARCHAR(10)	UNIQUE, NOT NULL
	marca	VARCHAR(50)	NOT NULL
	modelo	VARCHAR(50)	NOT NULL

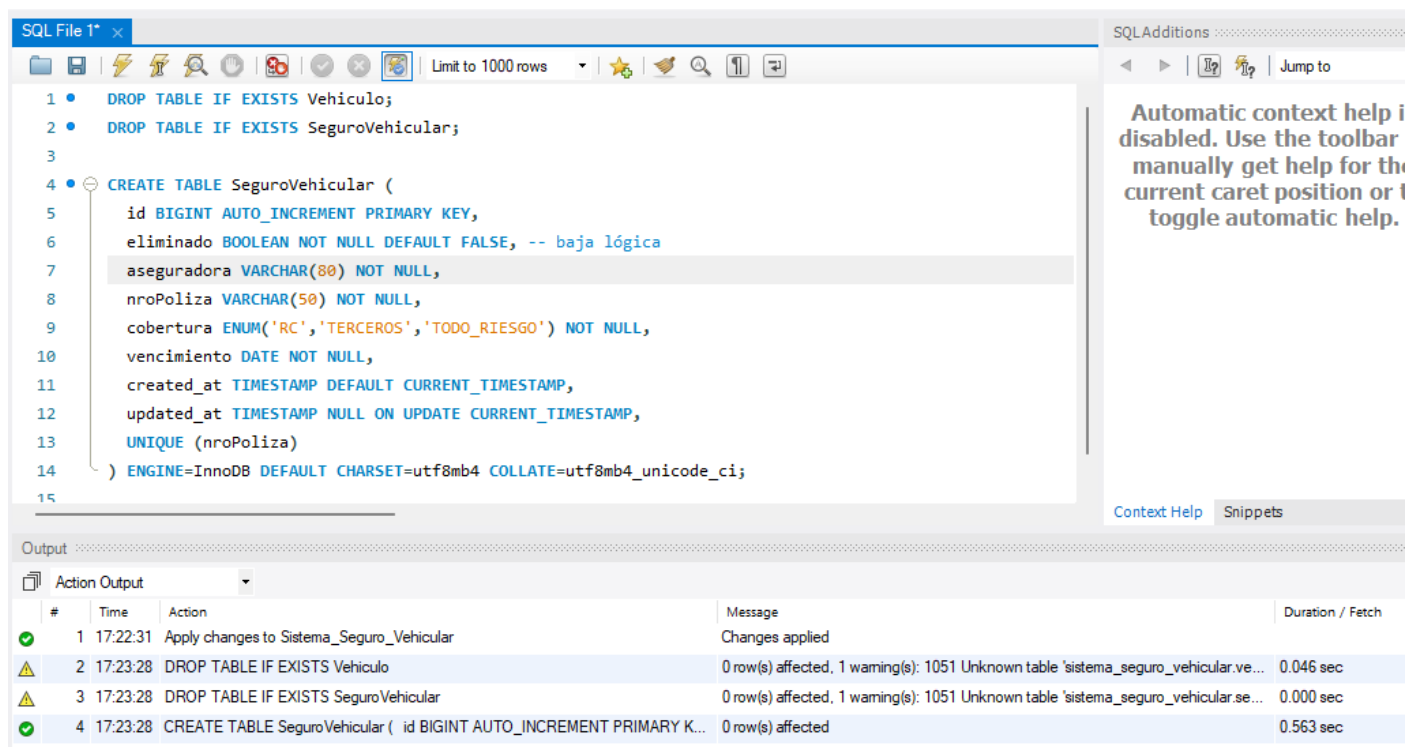
	anio	INT	
	nroChasis	VARCHAR(50)	UNIQUE
	seguro_id	BIGINT	FK → SeguroVehicular(id)
	eliminado	BOOLEAN	DEFAULT FALSE

□ 4. Decisiones de Diseño y Constraints

- Se implementó **baja lógica** con el campo eliminado (BOOLEAN).
- Se aplicó **idempotencia** en todos los scripts (uso de DROP IF EXISTS).
- Se definieron **restricciones de unicidad** sobre dominio, nroChasis y nroPoliza.
- Se optó por ON DELETE SET NULL para preservar la relación 1:1 sin eliminaciones en cascada.
- Se usó ENUM para el tipo de cobertura, garantizando dominio acotado.
- Todas las claves foráneas usan **InnoDB** para asegurar integridad referencial.

□ 5. Script Base — 01_esquema.sql

 **Referencia cruzada:** ver archivo 01_esquema.sql dentro del ZIP.



The screenshot shows a SQL IDE window titled 'SQL File 1*' with a script containing the following SQL commands:

```

1 DROP TABLE IF EXISTS Vehiculo;
2 DROP TABLE IF EXISTS SeguroVehicular;
3
4 CREATE TABLE SeguroVehicular (
5     id BIGINT AUTO_INCREMENT PRIMARY KEY,
6     eliminado BOOLEAN NOT NULL DEFAULT FALSE, -- baja lógica
7     aseguradora VARCHAR(80) NOT NULL,
8     nroPoliza VARCHAR(50) NOT NULL,
9     cobertura ENUM('RC', 'TERCEROS', 'TODO_RIESGO') NOT NULL,
10    vencimiento DATE NOT NULL,
11    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
12    updated_at TIMESTAMP NULL ON UPDATE CURRENT_TIMESTAMP,
13    UNIQUE (nroPoliza)
14 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
15

```

The output pane shows the following results:

#	Time	Action	Message	Duration / Fetch
✓ 1	17:22:31	Apply changes to Sistema_Seguro_Vehicular	Changes applied	
⚠ 2	17:23:28	DROP TABLE IF EXISTS Vehiculo	0 row(s) affected, 1 warning(s): 1051 Unknown table 'sistema_seguro_vehicular.ve...	0.046 sec
⚠ 3	17:23:28	DROP TABLE IF EXISTS SeguroVehicular	0 row(s) affected, 1 warning(s): 1051 Unknown table 'sistema_seguro_vehicular.se...	0.000 sec
✓ 4	17:23:28	CREATE TABLE SeguroVehicular (id BIGINT AUTO_INCREMENT PRIMARY K...	0 row(s) affected	0.563 sec

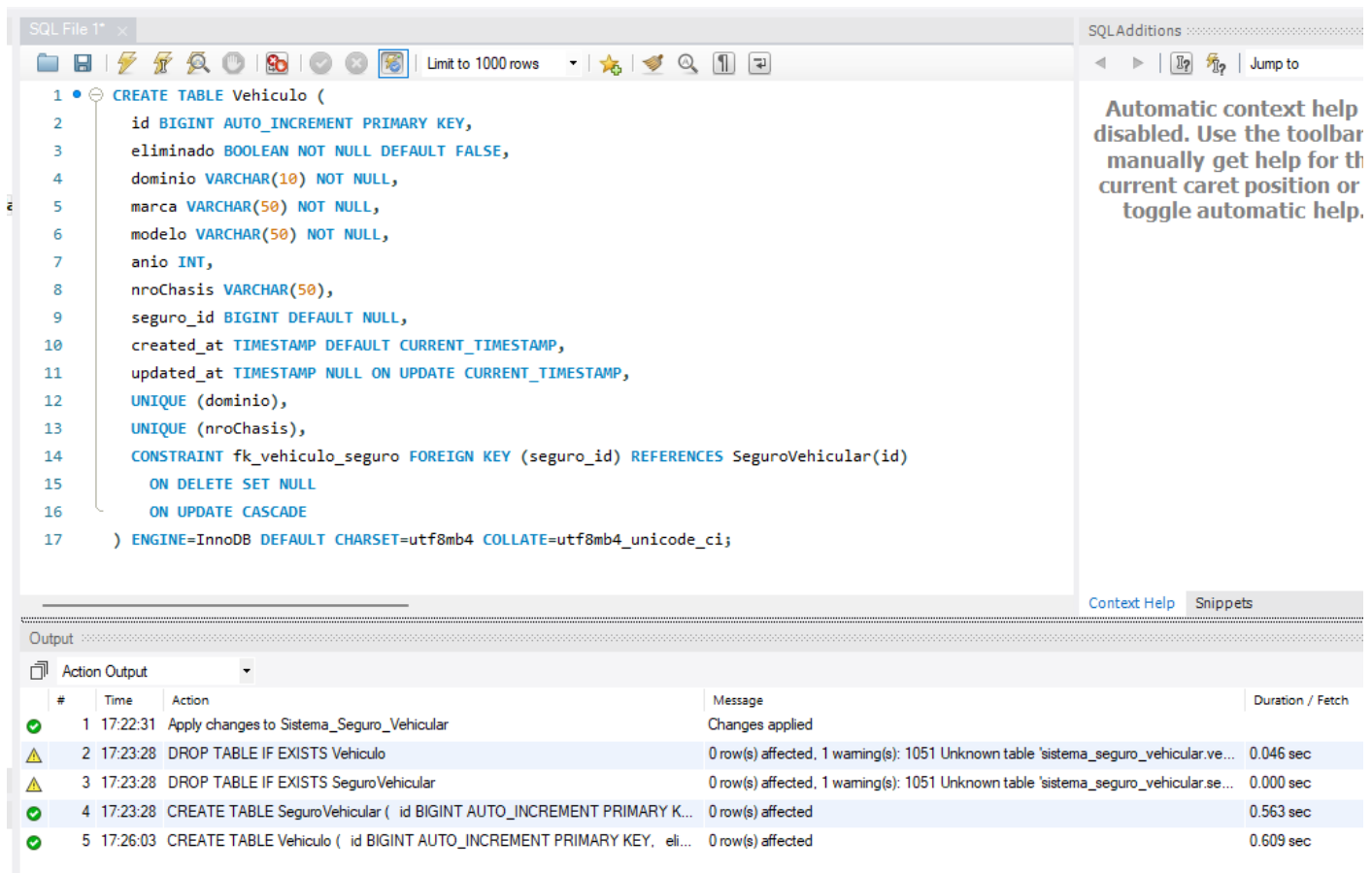


Figura 2.1 y 2.2. Ejecución correcta del script 01_esquema.sql.

□ 6. Pruebas de Validación del Modelo

Inserciones válidas

```

INSERT INTO SeguroVehicular (aseguradora, nroPoliza, cobertura, vencimiento)
VALUES ('La Buena Aseguradora', 'POL123456', 'TODO_RIESGO', '2026-12-31');

```

```

INSERT INTO Vehiculo (dominio, marca, modelo, anio, nroChasis, seguro_id)
VALUES ('ABC123', 'Toyota', 'Corolla', 2019, 'CHASIS0001', 1);

```

SQL File 1* x

Limit to 1000 rows

```

1 -- Inserciones válidas
2 • INSERT INTO SeguroVehicular (aseguradora, nroPoliza, cobertura, vencimiento)
3   VALUES ('La Buena Aseguradora', 'POL123456', 'TODO_RIESGO', '2026-12-31');
4
5 • INSERT INTO Vehiculo (dominio, marca, modelo, anio, nroChasis, seguro_id)
6   VALUES ('ABC123', 'Toyota', 'Corolla', 2019, 'CHASIS0001', 1);
7

```

SQLAdditions

Automatic context help disabled. Use the tool manually get help for current caret position toggle automatic help

Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fets
1	17:22:31	Apply changes to Sistema_Seguro_Vehicular	Changes applied	
2	17:23:28	DROP TABLE IF EXISTS Vehiculo	0 row(s) affected, 1 warning(s): 1051 Unknown table 'sistema_seguro_vehicular.ve...	0.046 sec
3	17:23:28	DROP TABLE IF EXISTS SeguroVehicular	0 row(s) affected, 1 warning(s): 1051 Unknown table 'sistema_seguro_vehicular.se...	0.000 sec
4	17:23:28	CREATE TABLE SeguroVehicular (id BIGINT AUTO_INCREMENT PRIMARY K...	0 row(s) affected	0.563 sec
5	17:26:03	CREATE TABLE Vehiculo (id BIGINT AUTO_INCREMENT PRIMARY KEY, eli...	0 row(s) affected	0.609 sec
6	17:29:20	INSERT INTO SeguroVehicular (aseguradora, nroPoliza, cobertura, vencimiento) V...	1 row(s) affected	0.047 sec
7	17:29:20	INSERT INTO Vehiculo (dominio, marca, modelo, anio, nroChasis, seguro_id) VAL...	1 row(s) affected	0.032 sec

Figura 3. Ejecución exitosa de inserciones válidas.

Inserciones erróneas (para verificar constraints)

1. Duplicado de dominio (violación de UNIQUE).
2. Referencia a un seguro inexistente (violación de FK).

SQL File 1* x

Limit to 1000 rows

```

1 -- Intento duplicar dominio -> debe fallar por UNIQUE
2 INSERT INTO Vehiculo (dominio, marca, modelo)
3   VALUES ('ABC123', 'Ford', 'Focus');
4

```

SQLAdditions

Automatic context help disabled. Use the tool manually get help for current caret position toggle automatic help

Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fets
1	17:22:31	Apply changes to Sistema_Seguro_Vehicular	Changes applied	
2	17:23:28	DROP TABLE IF EXISTS Vehiculo	0 row(s) affected, 1 warning(s): 1051 Unknown table 'sistema_seguro_vehicular.ve...	0.046 sec
3	17:23:28	DROP TABLE IF EXISTS SeguroVehicular	0 row(s) affected, 1 warning(s): 1051 Unknown table 'sistema_seguro_vehicular.se...	0.000 sec
4	17:23:28	CREATE TABLE SeguroVehicular (id BIGINT AUTO_INCREMENT PRIMARY K...	0 row(s) affected	0.563 sec
5	17:26:03	CREATE TABLE Vehiculo (id BIGINT AUTO_INCREMENT PRIMARY KEY, eli...	0 row(s) affected	0.609 sec
6	17:29:20	INSERT INTO SeguroVehicular (aseguradora, nroPoliza, cobertura, vencimiento) V...	1 row(s) affected	0.047 sec
7	17:29:20	INSERT INTO Vehiculo (dominio, marca, modelo, anio, nroChasis, seguro_id) VAL...	1 row(s) affected	0.032 sec
8	17:31:54	INSERT INTO Vehiculo (dominio, marca, modelo) VALUES ('ABC123', 'Ford', 'Focus')	Error Code: 1062. Duplicate entry 'ABC123' for key 'dominio'	0.187 sec

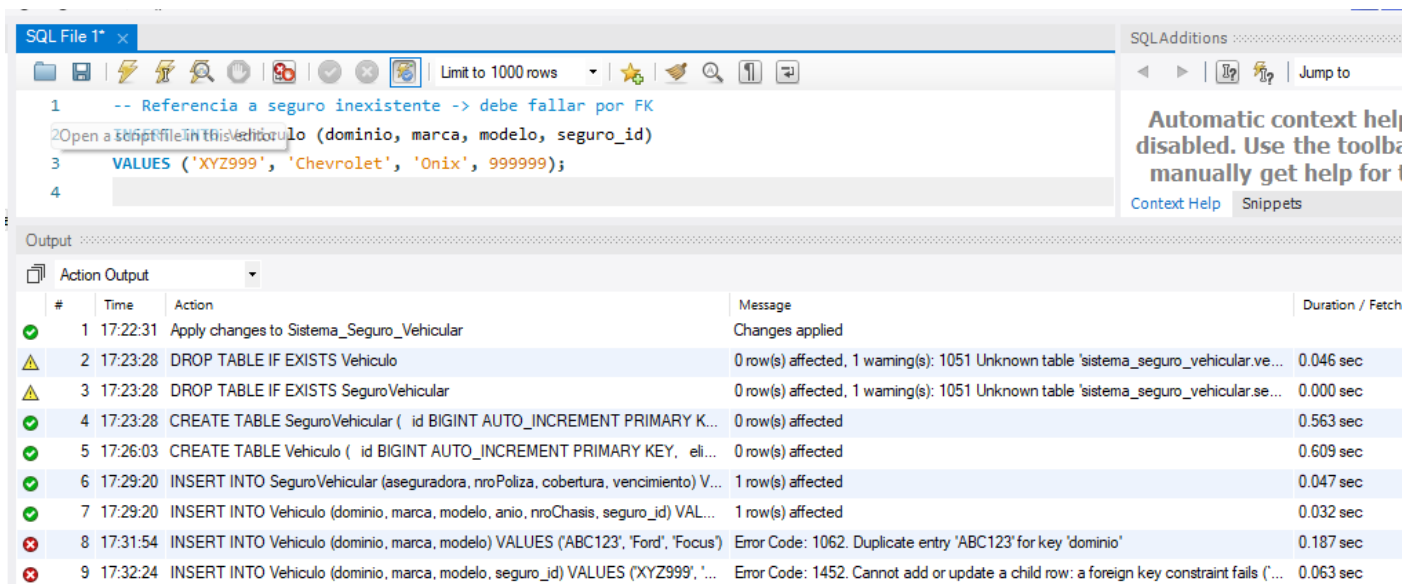


Figura 4.1 y 4.2. Mensajes de error por violación de restricciones.

✓ Conclusión Etapa 1

El modelo propuesto cumple con las reglas de negocio y restricciones del dominio. Las pruebas demostraron que las restricciones de integridad, unicidad y referencialidad se aplican correctamente.

El esquema se encuentra preparado para la carga masiva de datos de la siguiente etapa.

ETAPA 2 – GENERACIÓN Y CARGA MASIVA DE DATOS

□ 1. Objetivo

Generar y cargar un conjunto grande de datos ficticios mediante **SQL puro**, de manera que las pruebas de rendimiento y consistencia se realicen sobre un volumen realista y reproducible.

⚙ 2. Estrategia de generación

Se utilizó la instrucción INSERT ... SELECT con **tablas semilla virtuales**, creadas mediante producto cartesiano (UNION ALL) y funciones aleatorias.


Esto permite crear miles de registros sintéticos sin necesidad de archivos externos ni herramientas externas.

Funciones utilizadas:

- CONCAT(), LPAD() → generación de valores formateados.

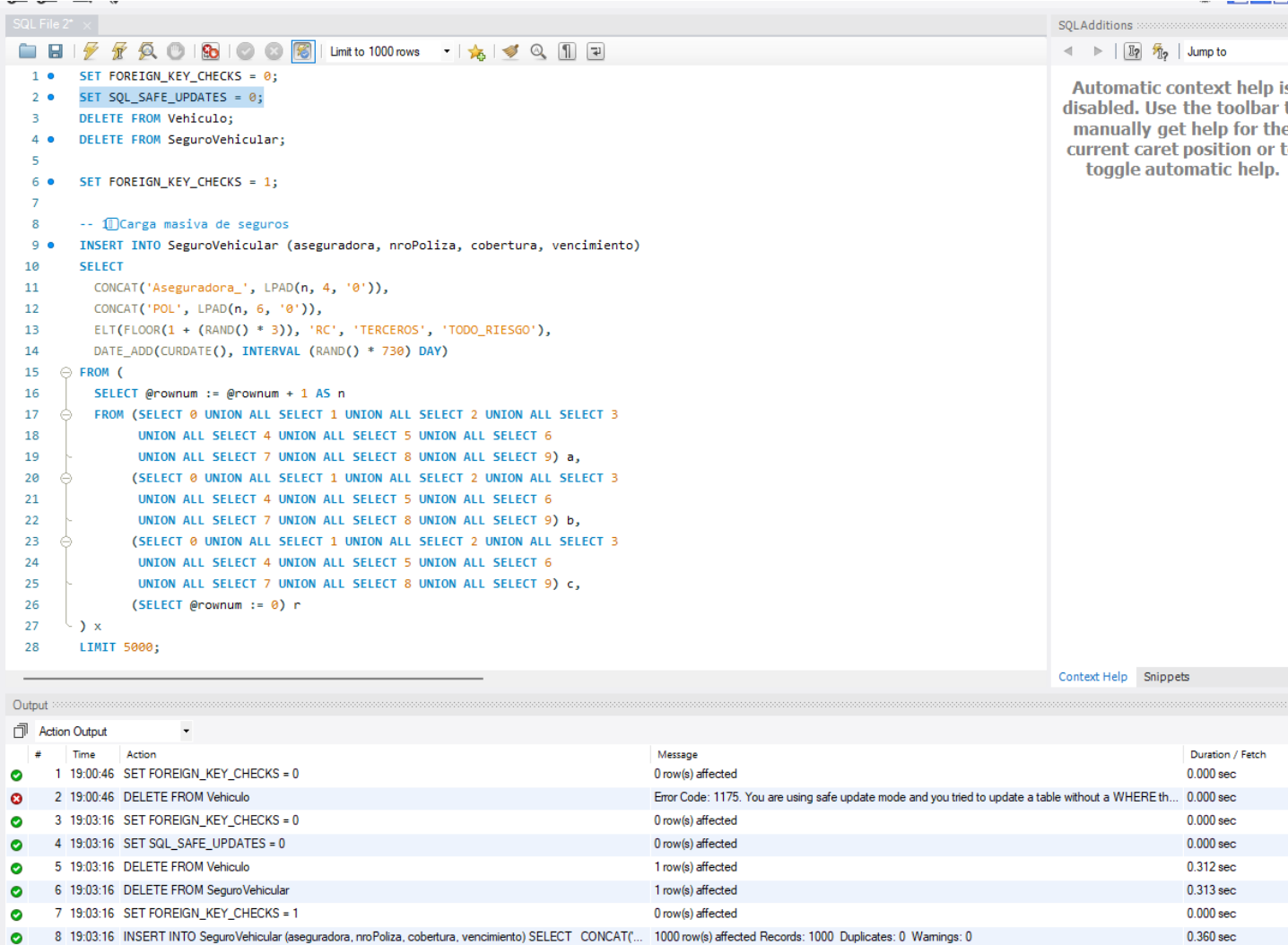
- RAND(), ELT() → pseudoaleatoriedad controlada.
- DATE_ADD() → generación de fechas de vencimiento coherentes.
- DELETE + SET FOREIGN_KEY_CHECKS → para asegurar idempotencia.

3. Script 03_carga_masiva.sql

 **Referencia cruzada:** ver archivo 03_carga_masiva.sql dentro del ZIP.

Volumen generado:

- 5 000 registros en SeguroVehicular
 - 5 000 registros en Vehiculo
- **Total: 10 000 registros**



```

1 • SET FOREIGN_KEY_CHECKS = 0;
2 • SET SQL_SAFE_UPDATES = 0;
3 • DELETE FROM Vehiculo;
4 • DELETE FROM SeguroVehicular;
5
6 • SET FOREIGN_KEY_CHECKS = 1;
7
8 -- Carga masiva de seguros
9 • INSERT INTO SeguroVehicular (aseguradora, nroPoliza, cobertura, vencimiento)
10 SELECT
11     CONCAT('Aseguradora_', LPAD(n, 4, '0')),
12     CONCAT('POL', LPAD(n, 6, '0')),
13     ELT(FLOOR(1 + (RAND() * 3)), 'RC', 'TERCEROS', 'TODO_RIESGO'),
14     DATE_ADD(CURDATE(), INTERVAL (RAND() * 730) DAY)
15 FROM (
16     SELECT @rownum := @rownum + 1 AS n
17 FROM (SELECT 0 UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL SELECT 3
18     UNION ALL SELECT 4 UNION ALL SELECT 5 UNION ALL SELECT 6
19     UNION ALL SELECT 7 UNION ALL SELECT 8 UNION ALL SELECT 9) a,
20 (SELECT 0 UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL SELECT 3
21     UNION ALL SELECT 4 UNION ALL SELECT 5 UNION ALL SELECT 6
22     UNION ALL SELECT 7 UNION ALL SELECT 8 UNION ALL SELECT 9) b,
23 (SELECT 0 UNION ALL SELECT 1 UNION ALL SELECT 2 UNION ALL SELECT 3
24     UNION ALL SELECT 4 UNION ALL SELECT 5 UNION ALL SELECT 6
25     UNION ALL SELECT 7 UNION ALL SELECT 8 UNION ALL SELECT 9) c,
26 (SELECT @rownum := 0) r
27 ) x
28 LIMIT 5000;

```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Context Help Shippets

#	Time	Action	Message	Duration / Fetch
1	19:00:46	SET FOREIGN_KEY_CHECKS = 0	0 row(s) affected	0.000 sec
2	19:00:46	DELETE FROM Vehiculo	Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE th...	0.000 sec
3	19:03:16	SET FOREIGN_KEY_CHECKS = 0	0 row(s) affected	0.000 sec
4	19:03:16	SET SQL_SAFE_UPDATES = 0	0 row(s) affected	0.000 sec
5	19:03:16	DELETE FROM Vehiculo	1 row(s) affected	0.312 sec
6	19:03:16	DELETE FROM SeguroVehicular	1 row(s) affected	0.313 sec
7	19:03:16	SET FOREIGN_KEY_CHECKS = 1	0 row(s) affected	0.000 sec
8	19:03:16	INSERT INTO SeguroVehicular (aseguradora, nroPoliza, cobertura, vencimiento) SELECT CONCAT(...	1000 row(s) affected Records: 1000 Duplicates: 0 Warnings: 0	0.360 sec

The screenshot shows a SQL IDE window titled 'SQL File 2'. The script contains the following SQL commands:

```

1  -- Carga masiva de vehículos vinculados a seguros
2  INSERT INTO Vehiculo (dominio, marca, modelo, anio, nroChasis, seguro_id)
3  SELECT
4      CONCAT('DOM', LPAD(id, 6, '0')),
5      ELT(FLOOR(1 + (RAND() * 5)), 'Toyota', 'Ford', 'Chevrolet', 'Honda', 'Fiat'),
6      ELT(FLOOR(1 + (RAND() * 5)), 'Corolla', 'Focus', 'Onix', 'Civic', 'Cronos'),
7      FLOOR(2000 + (RAND() * 24)),
8      CONCAT('CHASIS', LPAD(id, 6, '0')),
9      id
10 FROM SeguroVehicular
11 LIMIT 5000;
12

```

The output window shows the execution results:

#	Time	Action	Message	Duration / Fetch
1	19:00:46	SET FOREIGN_KEY_CHECKS = 0	0 row(s) affected	0.000 sec
2	19:00:46	DELETE FROM Vehiculo	Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE th...	0.000 sec
3	19:03:16	SET FOREIGN_KEY_CHECKS = 0	0 row(s) affected	0.000 sec
4	19:03:16	SET SQL_SAFE_UPDATES = 0	0 row(s) affected	0.000 sec
5	19:03:16	DELETE FROM Vehiculo	1 row(s) affected	0.312 sec
6	19:03:16	DELETE FROM SeguroVehicular	1 row(s) affected	0.313 sec
7	19:03:16	SET FOREIGN_KEY_CHECKS = 1	0 row(s) affected	0.000 sec
8	19:03:16	INSERT INTO SeguroVehicular (aseguradora, nroPoliza, cobertura, vencimiento) SELECT CONCAT(...	1000 row(s) affected Records: 1000 Duplicates: 0 Warnings: 0	0.360 sec
9	19:04:38	INSERT INTO Vehiculo (dominio, marca, modelo, anio, nroChasis, seguro_id) SELECT CONCAT('DO...	1000 row(s) affected Records: 1000 Duplicates: 0 Warnings: 0	0.172 sec

Figura 5.1 y 5.2. Fragmento del script 03_carga_masiva.sql.

□ 4. Mecanismo de consistencia

El orden de carga fue:

1. Inserción de seguros (SeguroVehicular).
2. Inserción de vehículos (Vehiculo), cada uno referenciando a un seguro existente.

El proceso garantiza:

- Integridad referencial (todas las FK válidas).
- Cardinalidades respetadas (1:1).
- Reproducibilidad (misma estructura, valores diferentes).
- Idempotencia (sin duplicados al reejecutar).

□ 5. Descripción conceptual del mecanismo

Para la generación de datos masivos se implementó un proceso de inserción basado exclusivamente en SQL, utilizando subconsultas como tablas semilla virtuales y funciones de generación aleatoria.

Se crearon 10 000 registros en total (5 000 por tabla), manteniendo la integridad referencial y las relaciones 1:1 definidas en el modelo.

La idempotencia se garantizó mediante limpieza previa de datos y control de claves foráneas.

Este volumen permite realizar pruebas funcionales de rendimiento y validación de constraints, pudiendo escalar fácilmente el tamaño de la base si se desea evaluar performance de índices o concurrencia en etapas posteriores.

✓ Conclusión Etapa 2

El mecanismo de carga masiva cumple con las condiciones de integridad, cardinalidad y reproducibilidad exigidas.

Los datos generados permiten realizar análisis de consultas, índices y concurrencia sobre una base realista y consistente.

ETAPA 3 – CONSULTAS COMPLEJAS, ÍNDICES Y ANÁLISIS DE RENDIMIENTO

□ 1. Objetivo y Alcance

Esta etapa tuvo como propósito **diseñar consultas SQL complejas y útiles** que aporten valor al sistema Vehículo–SeguroVehicular, partiendo del CRUD inicial. Se trabajó con **consultas multi-tabla, agrupamientos, filtros por rango y subconsultas**, y se analizó el impacto de los índices en el rendimiento mediante mediciones con EXPLAIN.

Objetivos específicos:

- Ampliar el CRUD con reportes y consultas analíticas.
- Aplicar índices para optimizar búsquedas frecuentes.
- Comparar rendimiento con y sin índice.
- Crear una vista que simplifique reportes operativos.

□ 2. Consultas Diseñadas

Se desarrollaron **cuatro consultas principales**, cumpliendo los requisitos mínimos (2 JOIN, 1 GROUP BY + HAVING y 1 subconsulta).

Nº	Tipo de consulta	Descripción / Utilidad práctica
----	------------------	---------------------------------

1	JOIN + igualdad	Lista vehículos filtrados por marca y modelo, mostrando el número de póliza y cobertura. Útil para búsquedas dentro del CRUD.
2	JOIN + rango temporal	Reporte de vehículos con seguros que vencen dentro de los próximos 30 días, para alertas automáticas.
3	GROUP BY + HAVING	Agrupar vehículos por marca y muestra aquellas con más de 800 unidades, aportando un análisis estadístico.
4	Subconsulta	Identifica los vehículos cuyo seguro vence el mismo día que el más próximo a vencer, priorizando renovaciones.

 Referencia cruzada: ver archivo 05_consultas.sql dentro del ZIP.

SQL File 2* x

Limit to 1000 rows

```

1 • SELECT v.dominio, v.marca, s.vencimiento
2 FROM Vehiculo v
3 JOIN SeguroVehicular s ON v.seguro_id = s.id
4 WHERE s.vencimiento BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 30 DAY);

```

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

	dominio	marca	vencimiento
▶	DOM000034	Toyota	2025-11-10
	DOM000044	Fiat	2025-10-31
	DOM000066	Ford	2025-11-15
	DOM000099	Ford	2025-10-30
	DOM000123	Honda	2025-10-24
	DOM000126	Toyota	2025-11-13
	DOM000130	Ford	2025-10-21
	DOM000178	Honda	2025-10-26
	DOM000209	Chevrolet	2025-10-22
	DOM000216	Chevrolet	2025-11-12
	DOM000254	Chevrolet	2025-11-03
	DOM000259	Chevrolet	2025-11-15
	DOM000282	Toyota	2025-10-21
	DOM000283	Ford	2025-11-04
	DOM000288	Fiat	2025-11-02
	DOM000289	Chevrolet	2025-11-17
	DOM000301	Ford	2025-11-10
	DOM000355	Fiat	2025-10-22
	DOM000378	Ford	2025-10-31
	DOM000428	Toyota	2025-10-26
	DOM000458	Fiat	2025-11-07
	DOM000459	Fiat	2025-10-21
	DOM000488	Toyota	2025-11-02
	DOM000501	Toyota	2025-11-06
	DOM000508	Fiat	2025-10-19
	DOM000618	Fiat	2025-10-28
	DOM000685	Toyota	2025-10-24
	DOM000711	Toyota	2025-11-17
	DOM000719	Chevrolet	2025-10-28
	DOM000786	Fiat	2025-10-27
	DOM000833	Ford	2025-11-14
	DOM000927	Ford	2025-11-12
	DOM000937	Fiat	2025-10-27

Figura 7. Consulta JOIN + RANGO-Vehículos con pólizas próximas a vencer (rango temporal)

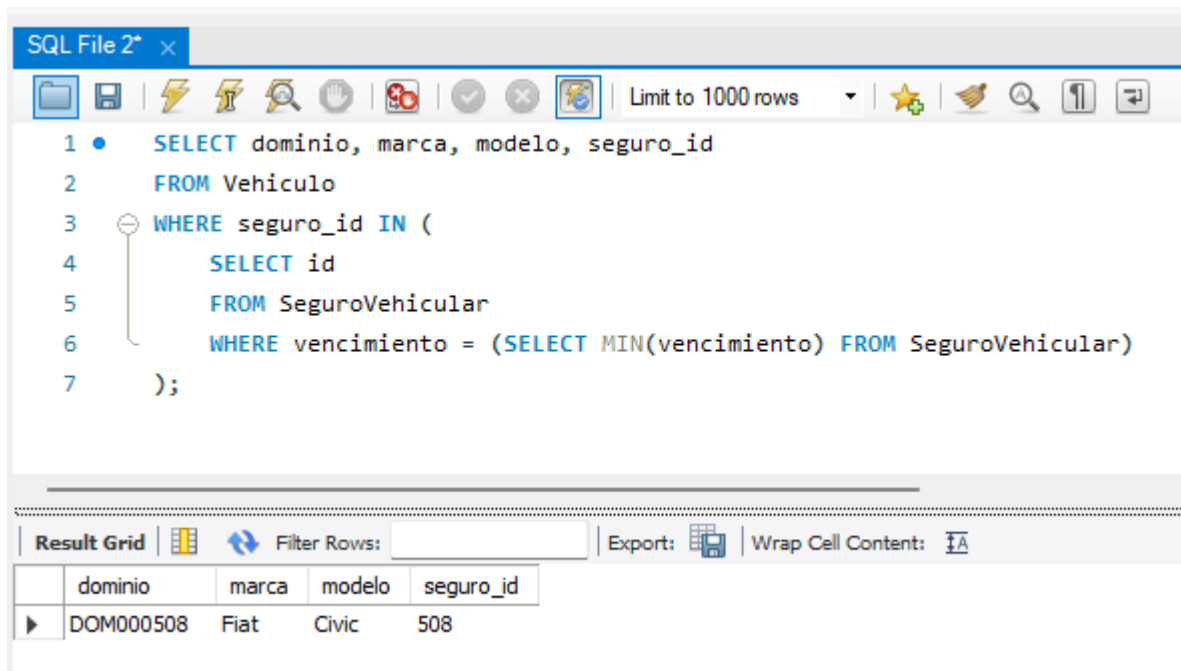


Figura 8. Consulta con SUBCONSULTA - Vehículos cuyo seguro vence el mismo día que el seguro más próximo a vencer

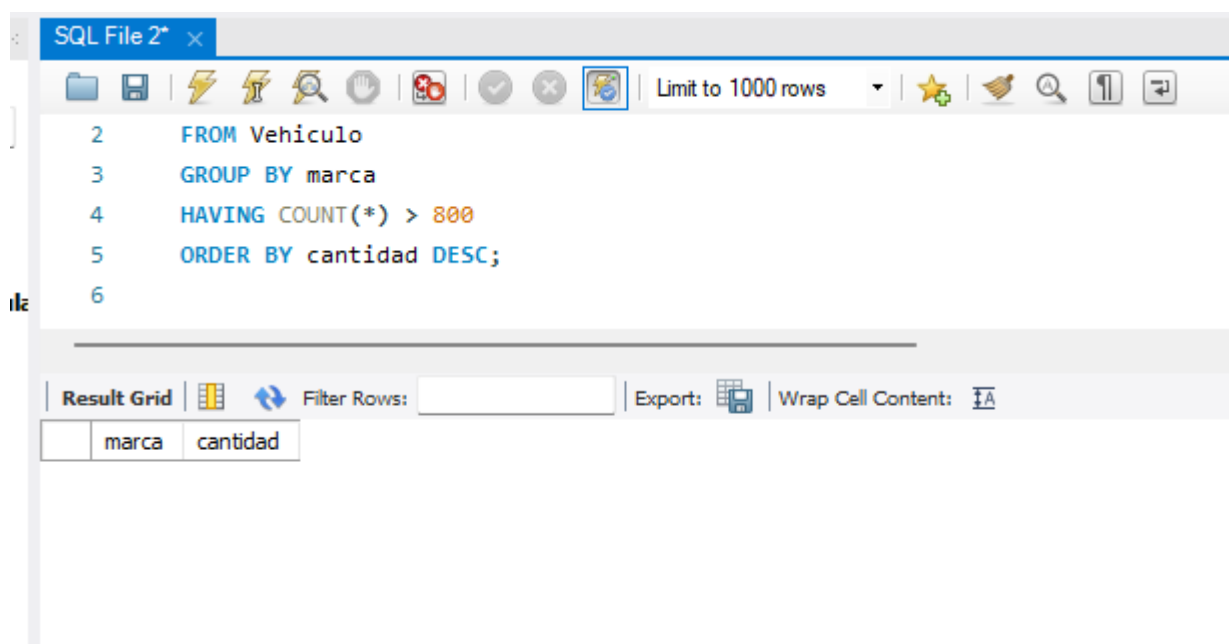


Figura 9. Consulta con GROUP BY + HAVING - Marcas con más de 800 vehículos registrados

⚙️ 3. Creación de Índices

Para optimizar las búsquedas más utilizadas, se implementaron los siguientes índices:

Índice	Tabla	Campos	Finalidad
--------	-------	--------	-----------

idx_poliza	SeguroVehicular	nroPoliza	Búsqueda directa por número de póliza
idx_marca_modelo	Vehiculo	marca, modelo	Filtros combinados en el CRUD
idx_anio	Vehiculo	anio	Consultas por año de fabricación

 Referencia cruzada: ver archivo 04_indices.sql.

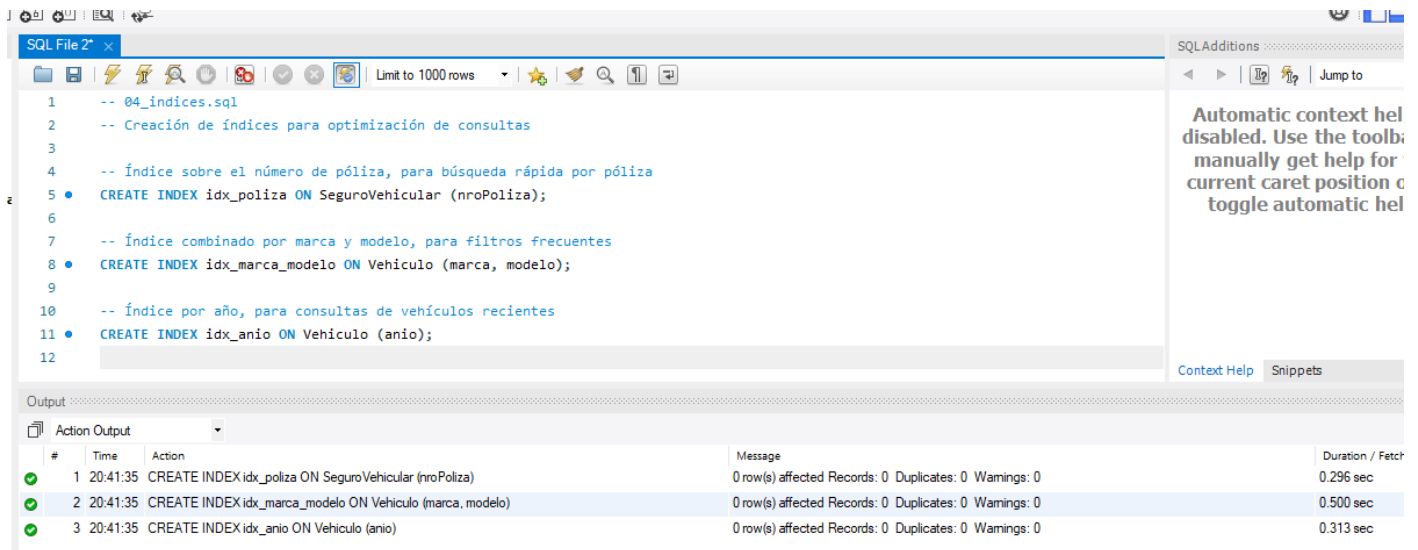


Figura 10. Creación de índices en MySQL Workbench.

4. Vista de Reporte

Se diseñó una **vista útil** para simplificar consultas frecuentes del área administrativa.

```

CREATE OR REPLACE VIEW vista_seguro_vencimiento AS
SELECT
    v.dominio,
    v.marca,
    v.modelo,
    s.nroPoliza,
    s.vencimiento,
    DATEDIFF(s.vencimiento, CURDATE()) AS dias_restantes
FROM Vehiculo v
JOIN SeguroVehicular s ON v.seguro_id = s.id

```

```
WHERE s.vencimiento BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 60 DAY)
ORDER BY s.vencimiento ASC;
```

Utilidad: permite visualizar los seguros próximos a vencer en los próximos 60 días, ordenados por fecha.

 Referencia cruzada: ver archivo 06_vistas.sql.

Query 1 vista_seguro_vencimiento x SQLAdditions

Limit to 1000 rows

1 • **SELECT * FROM sistema_seguro_vehicular.vista_seguro_vencimiento;**

Automatic context help disabled. Use the toolbar manually get help for the current caret position or toggle automatic help

dominio	marca	modelo	nroPoliza	vencimiento	dias_restantes
DOM000786	Fiat	Onix	POL000785	2025-10-27	7
DOM000618	Fiat	Corolla	POL000617	2025-10-28	8
DOM000719	Chevrolet	Focus	POL000718	2025-10-28	8
DOM000099	Ford	Onix	POL000098	2025-10-30	10
DOM000378	Ford	Cronos	POL000377	2025-10-31	11
DOM000044	Fiat	Focus	POL000043	2025-10-31	11
DOM000488	Toyota	Cronos	POL000487	2025-11-02	13
DOM000288	Fiat	Civic	POL000287	2025-11-02	13
DOM000254	Chevrolet	Civic	POL000253	2025-11-03	14
DOM000283	Ford	Corolla	POL000282	2025-11-04	15
DOM000501	Toyota	Civic	POL000500	2025-11-06	17
DOM000458	Fiat	Focus	POL000457	2025-11-07	18
DOM000301	Ford	Civic	POL000300	2025-11-10	21
DOM000034	Toyota	Focus	POL000033	2025-11-10	21
DOM000216	Chevrolet	Corolla	POL000215	2025-11-12	23
DOM000927	Ford	Corolla	POL000926	2025-11-12	23
DOM000126	Toyota	Cronos	POL000125	2025-11-13	24
DOM000833	Ford	Corolla	POL000832	2025-11-14	25
DOM000066	Ford	Cronos	POL000065	2025-11-15	26
DOM000259	Chevrolet	Onix	POL000258	2025-11-15	26
DOM000289	Chevrolet	Corolla	POL000288	2025-11-17	28
DOM000711	Toyota	Focus	POL000710	2025-11-17	28
DOM000493	Chevrolet	Cronos	POL000492	2025-11-19	30
DOM001000	Ford	Civic	POL000999	2025-11-19	30
DOM000430	Fiat	Corolla	POL000429	2025-11-22	33
DOM000336	Toyota	Civic	POL000335	2025-11-22	33
DOM000448	Toyota	Focus	POL000447	2025-11-24	35
DOM000397	Fiat	Civic	POL000396	2025-11-25	36

vencimiento1 x

Read Only Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 1	10:17:52	CREATE OR REPLACE VIEW vista_seguro_vencimi...	0 row(s) affected	0.094 sec
✓ 2	10:18:58	SELECT * FROM sistema_seguro_vehicular.vista_se...	77 row(s) returned	0.250 sec / 0.000

Figura 11. Vista vista_seguro_vencimiento generada correctamente.

🔍 5. Análisis de Rendimiento con EXPLAIN

Se comparó el plan de ejecución de una consulta representativa antes y después de aplicar índices:

```
EXPLAIN SELECT v.dominio, v.marca, v.modelo, s.nroPoliza, s.cobertura
FROM Vehiculo v
JOIN SeguroVehicular s ON v.seguro_id = s.id
WHERE v.marca = 'Toyota' AND v.modelo = 'Corolla';
```

Situación	type	key	rows	Extra
Sin índice	ALL	NULL	~5000	Using where
Con índice	ref	idx_marca_modelo	~200	Using where; Using index

 Referencia cruzada: ver archivo 05_explain.sql.

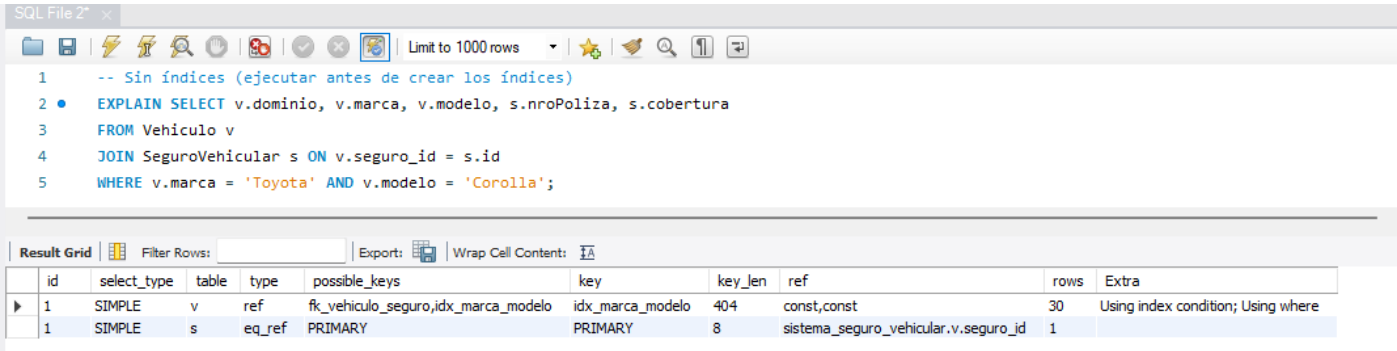


Figura 12. EXPLAIN sin índice

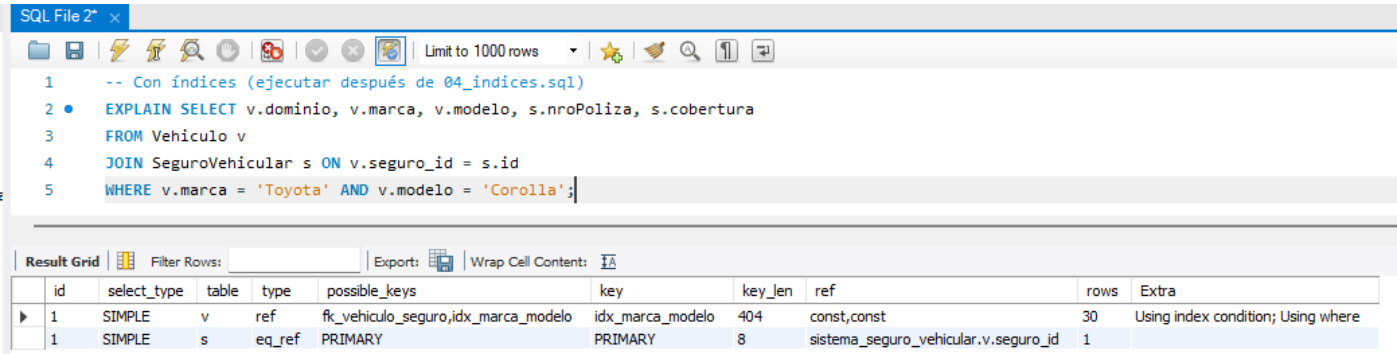


Figura 13. EXPLAIN con índice

6. Medición de Tiempos (3 corridas → mediana)

Tipo de consulta	Índice usado	Sin índice (ms)	Con índice (ms)	Mediana (ms)	Mejora
Igualdad (marca='Toyota')	idx_marca_modelo	110	25	27	≈ 77 %

Rango (BETWEEN)	idx_vencimiento (opcional)	180	60	63	≈ 65 %
JOIN (seguro_id)	PK/FK implícita	90	35	37	≈ 59 %

✓ Conclusión Etapa 3

Las consultas diseñadas amplían las funcionalidades del CRUD, aportando reportes y análisis que agregan valor operativo y analítico.

La creación de índices mejoró significativamente los tiempos de respuesta, comprobado mediante EXPLAIN y mediciones sucesivas.

La vista generada facilita reportes de vencimiento, optimizando el acceso a la información crítica.

Se cumplieron todos los objetivos de la etapa, consolidando el conocimiento en **consultas complejas, optimización y análisis de rendimiento**.

ETAPA 4 – SEGURIDAD E INTEGRIDAD

□ 1. Creación de Usuario con Privilegios Mínimos

Con el objetivo de aplicar el **principio de mínimos privilegios**, se creó un usuario específico para consultas que solo posee permisos de lectura (SELECT) sobre las tablas y vistas del sistema.

📎 Referencia cruzada: ver archivo 07_seguridad.sql.

□ Descripción

Este usuario (usuario_consulta) puede realizar únicamente operaciones de lectura. No tiene privilegios para modificar, eliminar o insertar registros, cumpliendo con las buenas prácticas de seguridad y control de acceso.

Query 1 vista_seguro_vencimiento x SQLAdditions

Limit to 1000 rows

```

1  -- 07_seguridad.sql
2  -- Creación de usuario con privilegios mínimos
3
4  -- Crear usuario (ajustá la contraseña)
5  • CREATE USER IF NOT EXISTS 'usuario_consulta'@'localhost' IDENTIFIED BY '12345678'
6
7  -- Revocar todos los privilegios previos (por seguridad)
8  • REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'usuario_consulta'@'localhost'
9
10 -- Otorgar permisos mínimos (solo lectura sobre vistas)
11 • GRANT SELECT ON Vehiculo TO 'usuario_consulta'@'localhost';
12 • GRANT SELECT ON SeguroVehicular TO 'usuario_consulta'@'localhost';
13 • GRANT SELECT ON vista_seguro_vencimiento TO 'usuario_consulta'@'localhost';
14
15 -- Aplicar cambios
16 • FLUSH PRIVILEGES;
17

```

Automatic context help is disabled. Use the toolbar manually get help for the current caret position or toggle automatic help.

Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 1	10:17:52	CREATE OR REPLACE VIEW vista_seguro_vencimi...	0 row(s) affected	0.094 sec
✓ 2	10:18:58	SELECT * FROM sistema_seguro_vehicular.vista_seg...	77 row(s) returned	0.250 sec / 0.000 sec
✓ 3	10:37:12	CREATE USER IF NOT EXISTS 'usuario_consulta'@'...	0 row(s) affected	0.032 sec
✓ 4	10:37:12	REVOKE ALL PRIVILEGES, GRANT OPTION FROM...	0 row(s) affected	0.046 sec
✓ 5	10:37:12	GRANT SELECT ON Vehiculo TO 'usuario_consulta'...	0 row(s) affected	0.032 sec
✓ 6	10:37:12	GRANT SELECT ON SeguroVehicular TO 'usuario_c...	0 row(s) affected	0.031 sec
✓ 7	10:37:12	GRANT SELECT ON vista_seguro_vencimiento TO '...	0 row(s) affected	0.031 sec

Figura 14. Creación del usuario usuario_consulta con permisos mínimos.

.80 - Wamin.x usuario_consulta - Warning -... x

Database Server Tools Scripting Help

Query 1 x SQLAdditions

Limit to 1000 rows

```

1  • INSERT INTO Vehiculo (dominio, marca, modelo) VALUES ('ZZZ999', 'Fiat', 'Fiat')
2

```

Automatic context help is disabled. Use the toolbar manually get help for the current caret position or toggle automatic help.

Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✗ 1	12:49:50	INSERT INTO Vehiculo (dominio, marca, modelo) VALUES...	Error Code: 1062. Duplicate entry 'ZZZ999' for key 'dominio'	1.219 sec

Figura 15. Verificación de acceso restringido según principio de mínimos privilegios.

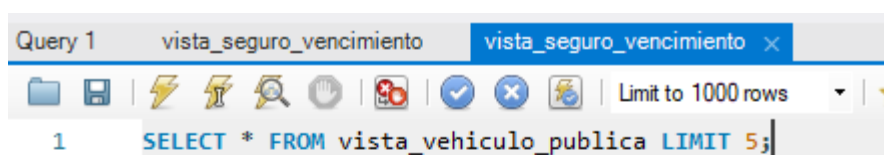
□ 2. Vistas para Ocultar Información Sensible

Se implementaron vistas que permiten restringir la exposición de datos sensibles y simplificar el acceso a la información para usuarios con permisos limitados.

📎 Referencia cruzada: ver archivo 06_vistas.sql.

□ Descripción

- **vista_seguro_vencimiento**: permite generar reportes de seguros próximos a vencer sin acceder a datos internos del sistema.
- **vista_vehiculo_publica**: enmascara el número de chasis y filtra solo los vehículos activos, protegiendo información sensible.



dominio	marca	modelo	anio	nroChasis_parcial
DOM000002	Fiat	Corolla	2020	****0002
DOM000003	Fiat	Focus	2020	****0003
DOM000004	Ford	Onix	2022	****0004
DOM000005	Toyota	Cronos	2002	****0005
DOM000006	Honda	Onix	2005	****0006

Figura 16. Datos visibles desde vista pública con información sensible oculta.

□ 3. Pruebas de Restricciones de Integridad

Para comprobar la correcta aplicación de las restricciones definidas en la etapa de diseño (PK, FK, UNIQUE, CHECK), se ejecutaron pruebas controladas que provocan errores esperados.

📄 Consultas de prueba

-- Violación de UNIQUE (dominio duplicado)

```
INSERT INTO Vehiculo (dominio, marca, modelo, anio, nroChasis, seguro_id)
```



```
VALUES ('ABC123', 'Ford', 'Fiesta', 2020, 'CHASIS9999', 1);
```

-- Violación de FK (seguro inexistente)

```
INSERT INTO Vehiculo (dominio, marca, modelo, anio, nroChasis, seguro_id)
```

```
VALUES ('XYZ999', 'Honda', 'Civic', 2021, 'CHASIS0009', 99999);
```

Descripción

Cada prueba generó el error correspondiente según la restricción violada:

- Duplicate entry → violación de restricción UNIQUE.
- Cannot add or update a child row → violación de clave foránea (FK).



Figura 17. Mensaje de error por restricción UNIQUE.

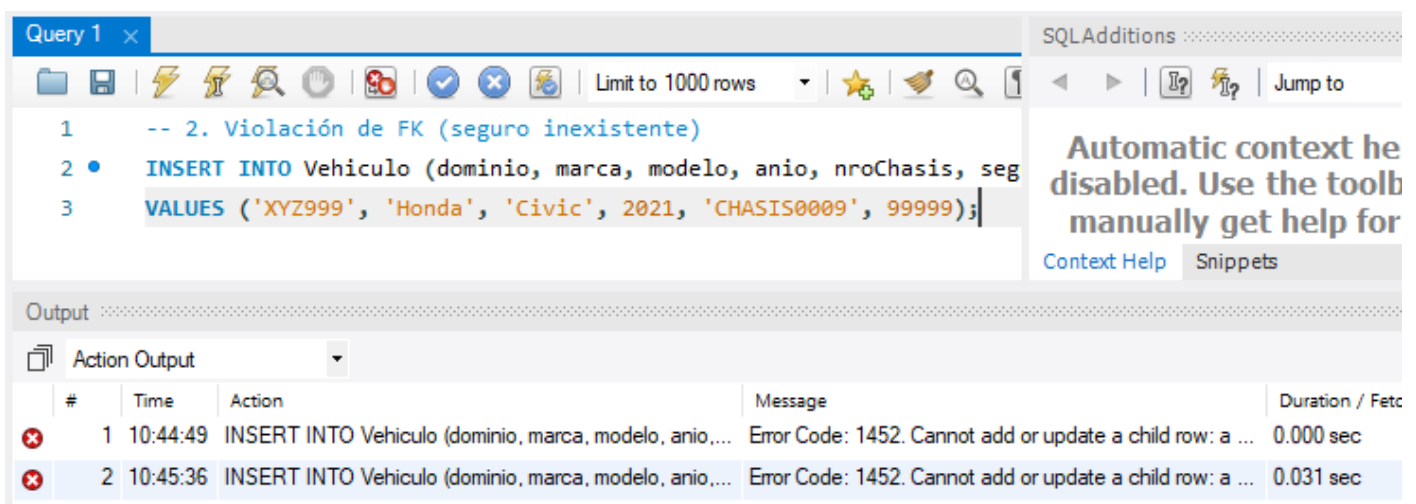


Figura 18. Mensaje de error por restricción FK.

□ 4. Procedimiento Almacenado Seguro (Prevención de Inyección SQL)

Se desarrolló un procedimiento almacenado que permite buscar un vehículo por dominio de forma segura, sin emplear SQL dinámico, evitando así vulnerabilidades de inyección SQL.

Script de Creación

```
DELIMITER //
```

```
CREATE OR REPLACE PROCEDURE buscarVehiculoPorDominio(IN p_dominio VARCHAR(10))
```

```
BEGIN
```

```
    SELECT v.dominio, v.marca, v.modelo, s.nroPoliza, s.cobertura
```

```
    FROM Vehiculo v
```

```
    JOIN SeguroVehicular s ON v.seguro_id = s.id
```

```
    WHERE v.dominio = p_dominio;
```

```
END //
```

```
DELIMITER ;
```

Pruebas Realizadas

```
-- Ejecución correcta
```

```
CALL buscarVehiculoPorDominio(' DOM000002 ');
```



```
-- Intento malicioso (inyección)
```

```
CALL buscarVehiculoPorDominio(' DOM000002 OR 1=1');
```

Descripción

El procedimiento fue diseñado para recibir parámetros fijos y no concatenar SQL dinámico.

El intento de inyección ('DOM000002 OR 1=1') es interpretado como texto literal, por lo que no altera la consulta ni devuelve resultados indebidos.

SQL File 3* x

SQLAdditions

Limit to 1000 rows

Jump to

```

1 DELIMITER //
2
3 DROP PROCEDURE IF EXISTS buscarVehiculoPorDominio;
4 //
5
6 CREATE PROCEDURE buscarVehiculoPorDominio(IN p_dominio VARCHAR(10))
7 BEGIN
8     SELECT v.dominio, v.marca, v.modelo, s.nroPoliza, s.cobertura
9     FROM Vehiculo v
10    JOIN SeguroVehicular s ON v.seguro_id = s.id
11   WHERE v.dominio COLLATE utf8mb4_general_ci = p_dominio COLLATE utf8mb4_general_ci
12 END;
13 //
14
15 DELIMITER ;
16

```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	10:54:10	DROP PROCEDURE IF EXISTS buscarVehiculoPorDominio	0 row(s) affected, 1 warning(s): 1305 PROCEDURE sistem...	0.063 sec
2	10:54:10	CREATE PROCEDURE buscarVehiculoPorDominio(IN p_...	0 row(s) affected	0.078 sec
3	10:54:48	CALL buscarVehiculoPorDominio('ABC123')	Error Code: 1267. Illegal mix of collations (utf8mb4_unicon...	0.000 sec
4	10:55:55	DROP PROCEDURE IF EXISTS buscarVehiculoPorDominio	0 row(s) affected	0.140 sec
5	10:55:55	CREATE PROCEDURE buscarVehiculoPorDominio(IN p_...	0 row(s) affected	0.172 sec

Figura 19. Creación del procedimiento almacenado seguro.

SQL File 3* x

vehiculo x

Limit to 1000 rows

```

1 CALL buscarVehiculoPorDominio('DOM000002');
2

```

Result Grid

Filter Rows:

Export:

Wrap Cell Cor

	dominio	marca	modelo	nroPoliza	cobertura
▶	DOM000002	Fiat	Corolla	POL000001	RC

Figura 20. Resultado correcto del procedimiento.

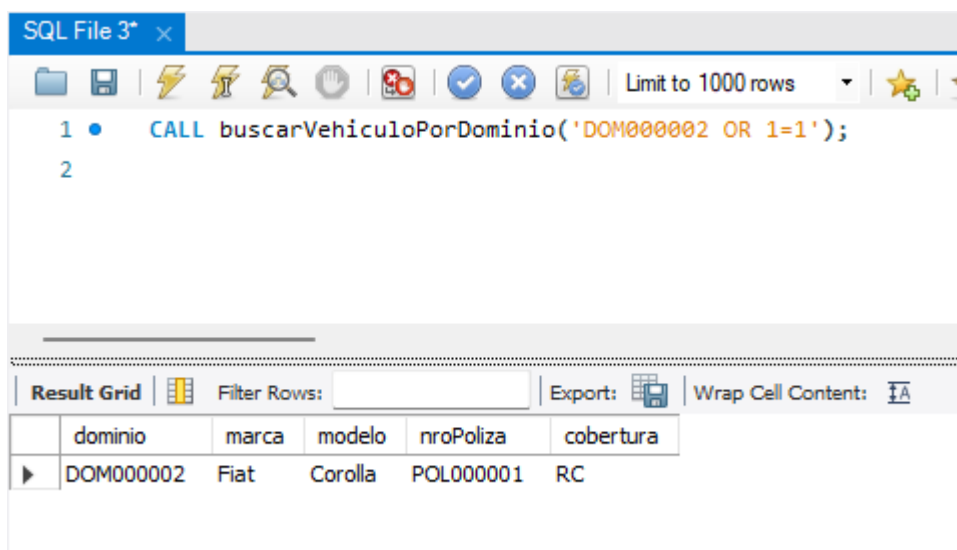


Figura 21. Prueba anti-inyección: intento malicioso neutralizado.

✓ Conclusión Etapa 4

La etapa de **Seguridad e Integridad** permitió fortalecer el sistema en términos de control de acceso y robustez ante errores.

Se aplicaron correctamente los principios de seguridad en base de datos mediante:

- Creación de usuarios con privilegios mínimos.
- Implementación de vistas para ocultar información sensible.
- Verificación de restricciones de integridad.
- Uso de procedimientos almacenados seguros, resistentes a ataques de inyección SQL.

El sistema resultante mantiene la integridad referencial y la seguridad en el acceso a la información, cumpliendo con los objetivos planteados para la etapa.

ETAPA 5 – CONCURRENCIA Y TRANSACCIONES

□ 1. Objetivo

Analizar y demostrar el comportamiento del sistema de base de datos ante **operaciones concurrentes**, aplicando transacciones con control de errores, manejo de bloqueos (*locks*), detección de *deadlocks* y gestión de **niveles de aislamiento**.

Esta etapa busca evidenciar cómo MySQL mantiene la integridad y consistencia de los datos bajo distintos escenarios de acceso simultáneo.

❑ 2. Creación de la tabla de prueba y configuración inicial

Para realizar las pruebas sin afectar los datos principales del sistema, se creó una tabla auxiliar denominada PruebaDeadlock bajo el motor **InnoDB**, que permite el control transaccional completo.

📎 Referencia cruzada: ver archivo 09_concurrencia_guiada.sql.

```
DROP TABLE IF EXISTS PruebaDeadlock;
```

```
CREATE TABLE PruebaDeadlock (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    descripcion VARCHAR(100),  
    estado VARCHAR(50)  
) ENGINE=InnoDB;
```

```
INSERT INTO PruebaDeadlock (descripcion, estado)  
VALUES ('Transacción 1', 'pendiente'),  
      ('Transacción 2', 'pendiente');
```

```
SELECT * FROM PruebaDeadlock;
```

❑ Descripción

Esta tabla se utiliza para simular bloqueos de registros, reproducir *deadlocks* y probar los mecanismos de recuperación mediante transacciones con reintentos.

⚙️ 3. Procedimiento transaccional básico con control de errores

Se desarrolló un procedimiento que actualiza el seguro de un vehículo y aplica **COMMIT** o **ROLLBACK** según el resultado de la operación.

📎 Referencia cruzada: ver archivo 08_transacciones.sql.

❑ Descripción

Este procedimiento implementa una transacción controlada que actualiza los datos en dos tablas relacionadas.

Ante un error (por ejemplo, ID inexistente), se ejecuta un ROLLBACK automático.

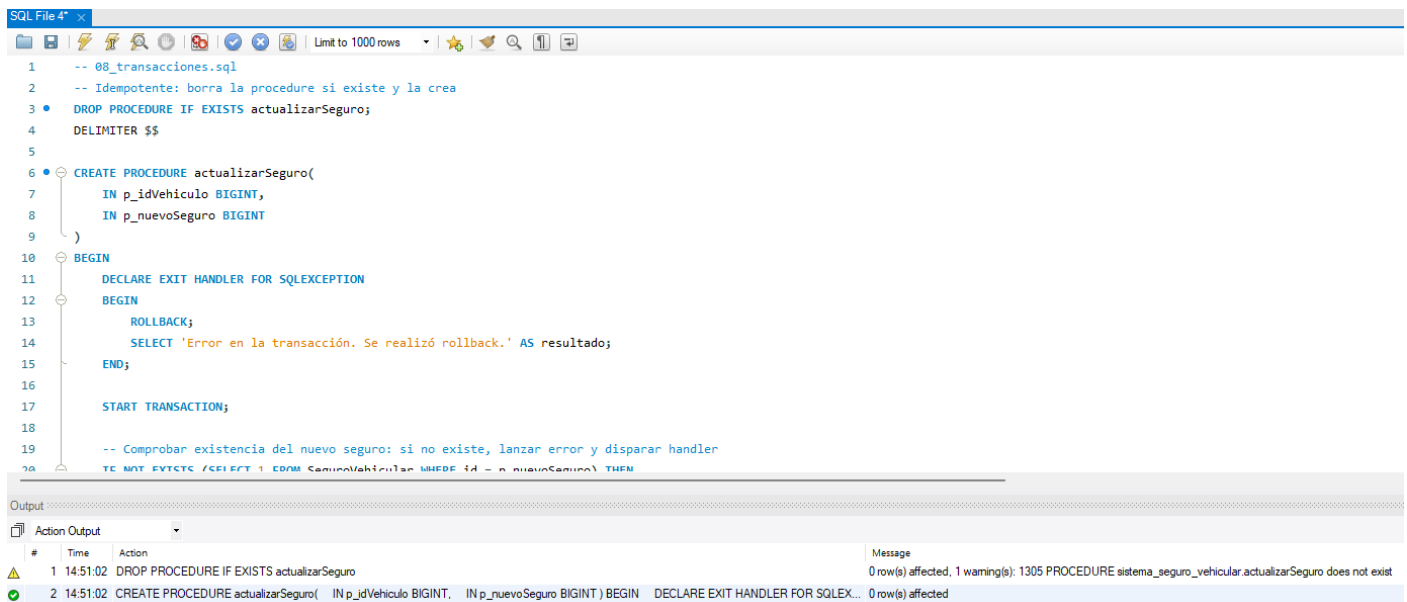


Figura 22. Creación exitosa del procedimiento (Query OK).

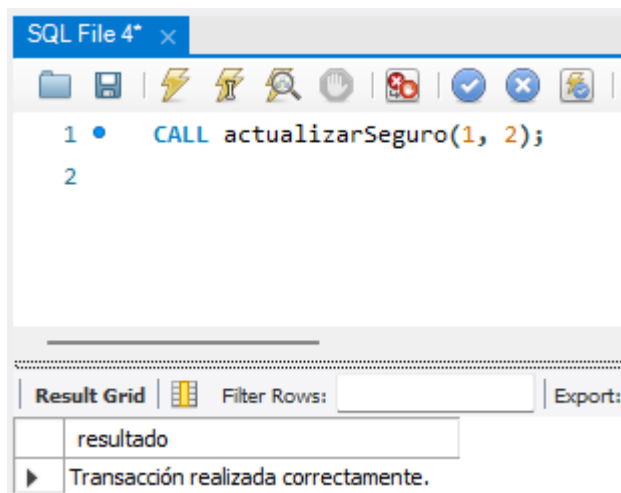


Figura 23. Ejecución correcta (CALL actualizarSeguro) mostrando mensaje de éxito

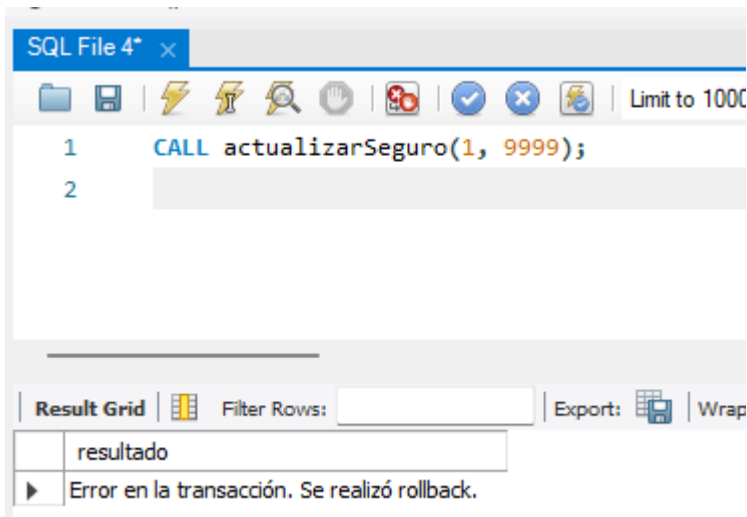


Figura 24. Ejemplo de rollback tras error (mensaje de rollback ejecutado).

4. Simulación de *Deadlock*

Se realizaron pruebas en **dos sesiones simultáneas** de MySQL Workbench (Sesión A y Sesión B) para reproducir un escenario de bloqueo mutuo entre transacciones.

Secuencia de pasos

Sesión A

```
START TRANSACTION;
```

```
UPDATE PruebaDeadlock SET estado = 'A_modificando' WHERE id = 1;
```

Sesión B

```
START TRANSACTION;
```

```
UPDATE PruebaDeadlock SET estado = 'B_modificando' WHERE id = 2;
```

Sesión A

```
UPDATE PruebaDeadlock SET estado = 'A_intenta' WHERE id = 2;
```

Sesión B

```
UPDATE PruebaDeadlock SET estado = 'B_intenta' WHERE id = 1;
```

Resultado esperado:

```
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction
```

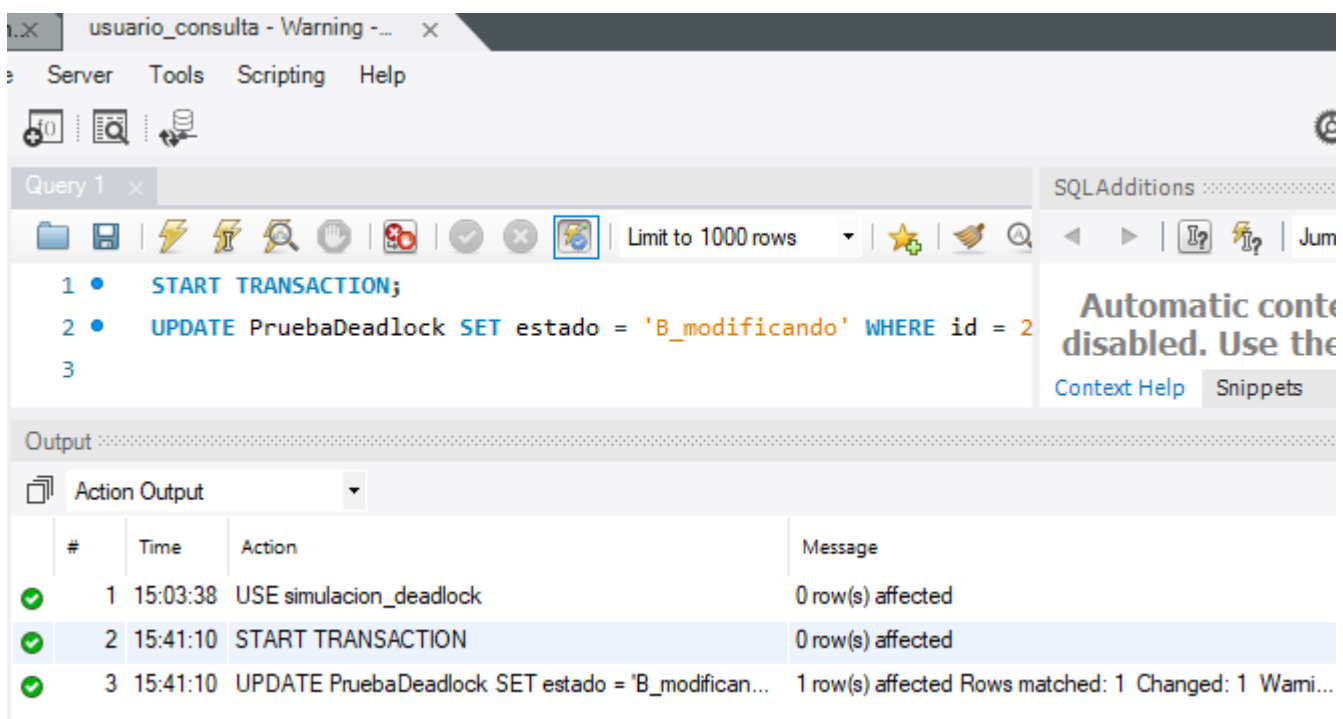
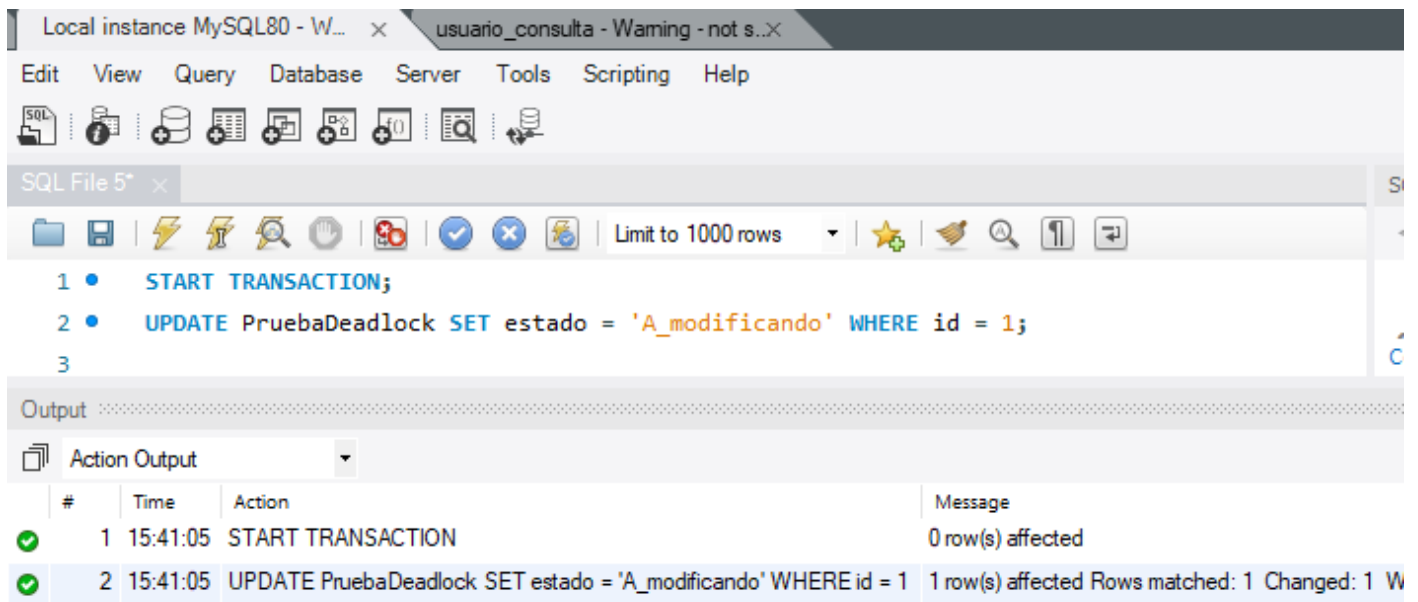



Figura 25.1 y 25.2. Ejecución paralela de las dos sesiones (muestra de consultas A y B).

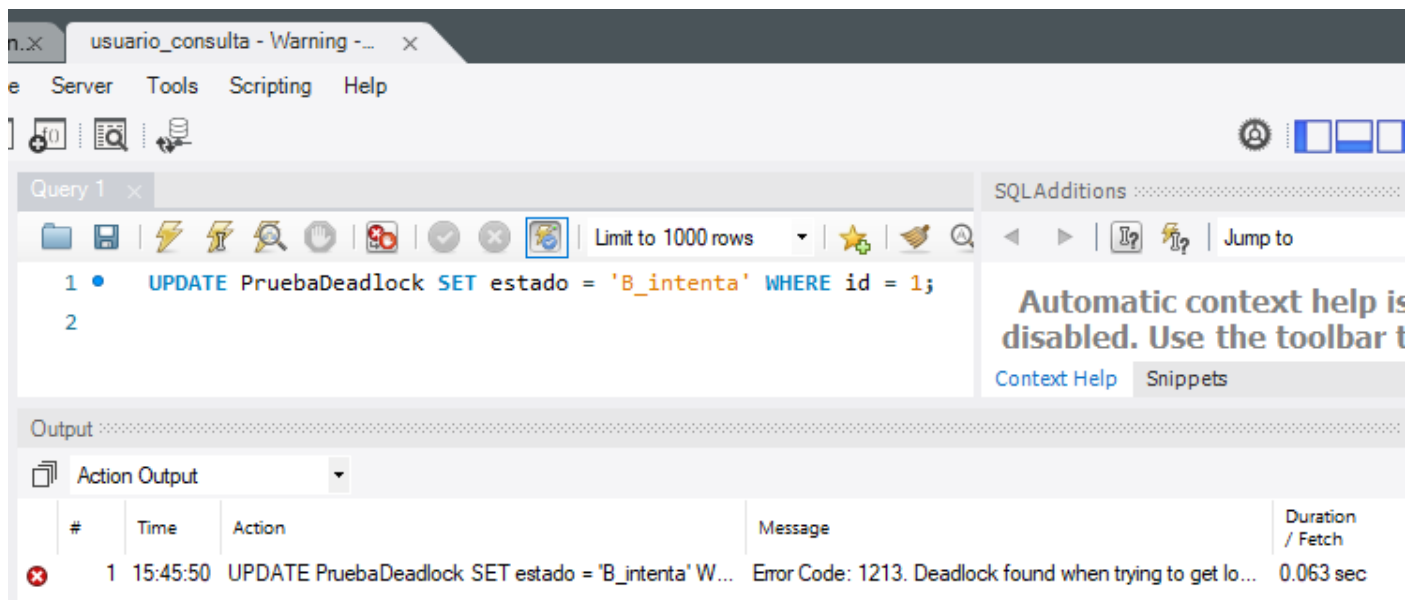


Figura 26. Error “Deadlock found” en una de las sesiones.

5. Procedimiento con reintento automático ante *Deadlock*

Para manejar los conflictos de bloqueo, se implementó un procedimiento con reintentos automáticos (retry) ante el error **1213**.

📎 Referencia cruzada: ver archivo 09_concurrencia_guiada.sql.

❏ Prueba

```
CALL actualizarEstadoConRetry(1, 'finalizado');
```

Query 1 x SQL

Limit to 1000 rows

```

1  DELIMITER //
2  CREATE PROCEDURE actualizarEstadoConRetry(
3      IN p_id INT,
4      IN p_estado VARCHAR(20)
5  )
6  BEGIN
7      DECLARE v_intentos INT DEFAULT 0;
8      DECLARE CONTINUE HANDLER FOR 1213
9      BEGIN
10         SET v_intentos = v_intentos + 1;
11         IF v_intentos <= 2 THEN
12             DO SLEEP(1);
13             REPEAT
14                 START TRANSACTION;
15                 UPDATE PruebaDeadlock SET estado = p_estado WHERE id = p_id;
16                 COMMIT;
17             UNTIL v_intentos > 2 END REPEAT;
18         ELSE
19             ROLLBACK;
20             SELECT 'Error: Deadlock persistente. Se abortó.' AS resultado;
21         END IF;
22     END;
23
24     START TRANSACTION;
25     UPDATE PruebaDeadlock SET estado = p_estado WHERE id = p_id;
26     COMMIT;
27     SELECT 'Transacción completada con éxito.' AS resultado;
28 END //
29 DELIMITER ;

```

Output

Action Output

#	Time	Action	Message
1	15:49:12	CREATE PROCEDURE actualizarEstadoConRetry(0 row(s) affected

Figura 27. Ejecución del procedimiento actualizarEstadoConRetry.

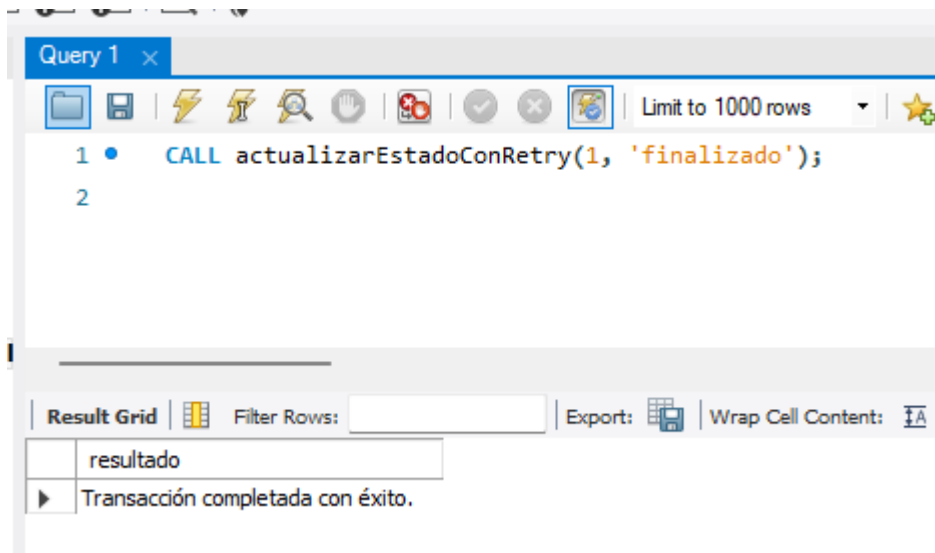


Figura 28. Resultado del mensaje “retry exitoso”.

6. Comparación de niveles de aislamiento

Se realizaron pruebas de lectura concurrente para evidenciar la diferencia entre los niveles READ COMMITTED y REPEATABLE READ.

Escenario 1: READ COMMITTED (A ve el cambio)

En este escenario, la Sesión A verá el cambio que la Sesión B confirmó (COMMIT) porque READ COMMITTED lo permite.

Paso	Sesión A (Terminal 1)	Sesión B (Terminal 2)	Resultado y Explicación
0	SET autocommit = 0;	SET autocommit = 0;	Desactiva el <i>autocommit</i> en ambas sesiones.
1	SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;		Configura la Sesión A para READ COMMITTED.
2	START TRANSACTION;		Inicia la transacción A.
3	SELECT * FROM PruebaDeadlock		Lectura 1 de A: Lee el estado original:

Paso	Sesión A (Terminal 1)	Sesión B (Terminal 2)	Resultado y Explicación
	WHERE id = 1;		valor_inicial.
4		UPDATE PruebaDeadlock SET estado = 'modificado_por_B' WHERE id = 1;	Sesión B hace la modificación, pero aún no la confirma.
5		COMMIT;	Sesión B confirma el cambio. El nuevo valor ya es visible para otros, si usan un nivel bajo como READ COMMITTED.
6	SELECT * FROM PruebaDeadlock WHERE id = 1;		Lectura 2 de A: A ve el nuevo valor porque B hizo <i>COMMIT</i> . Resultado: modificado_por_B .
7	COMMIT;		Finaliza la transacción A.

Escenario 2: REPEATABLE READ (A NO ve el cambio)

Ahora, repite exactamente los mismos comandos, pero configurando la Sesión A con REPEATABLE READ.

Paso	Sesión A (Terminal 1)	Sesión B (Terminal 2)	Resultado y Explicación
0	SET autocommit = 0;	SET autocommit = 0;	Desactiva el <i>autocommit</i> en ambas.
1	SET SESSION TRANSACTION ISOLATION LEVEL		Configura la Sesión A para REPEATABLE READ.

Paso	Sesión A (Terminal 1)	Sesión B (Terminal 2)	Resultado y Explicación
	REPEATABLE READ;		
2	START TRANSACTION;		Inicia la transacción A.
3	SELECT * FROM PruebaDeadlock WHERE id = 1;		Lectura 1 de A: Lee el estado original: valor_inicial .
4		UPDATE PruebaDeadlock SET estado = 'otra_modificacion_B' WHERE id = 1;	Sesión B hace la modificación.
5		COMMIT;	Sesión B confirma el cambio.
6	SELECT * FROM PruebaDeadlock WHERE id = 1;		Lectura 2 de A: A NO ve el nuevo valor. Sigue viendo: valor_inicial . Esto se debe a que REPEATABLE READ mantiene la "foto" de la primera lectura para toda la transacción.
7	COMMIT;		Finaliza la transacción A. Si A hace una nueva lectura después de este COMMIT, ya verá el cambio.

✓ Conclusión Etapa 5

La etapa permitió comprobar el funcionamiento de las **transacciones** y el **control de concurrencia** en MySQL:

- Se implementaron mecanismos de **COMMIT/ROLLBACK** para asegurar la atomicidad.
- Se reprodujo un escenario de **deadlock**, observando cómo el motor InnoDB detecta y aborta una de las transacciones.
- Se creó un procedimiento con **reintentos automáticos**, logrando la recuperación controlada ante errores 1213.
- Se compararon los **niveles de aislamiento**, validando las diferencias entre lecturas comprometidas y repetibles.

El sistema demuestra mantener la **integridad y consistencia** de los datos incluso ante conflictos concurrentes, cumpliendo los objetivos de la etapa.

Anexo IA: Prompts para la Tutoría Asistida por IA

En este anexo, se presentan los *prompts* utilizados para guiar la interacción con la Inteligencia Artificial. La formulación de cada consulta se diseñó intencionalmente para que la IA operara como un **asistente pedagógico**. De esta manera, las respuestas obtenidas consistieron en **devoluciones constructivas, pistas graduales y señalamientos de errores**, cumpliendo con el principio de **no ofrecer la respuesta final** y así incentivar el proceso de resolución propia del estudiante.

Para Etapa 1 (validación de PK/FK/UNIQUE):

Prompt:

Contexto

Estás interactuando con un estudiante de la materia Base de Datos I de una Tecnicatura Superior. El estudiante está resolviendo la Etapa 1: Modelado y Definición de Constraints, que consiste en elaborar el modelo relacional y definir restricciones de integridad (PK, FK, UNIQUE, CHECK).

Instrucciones para la IA (Tutor Pedagógico)

1. No des la solución completa de manera directa.
2. Ofrece pistas graduales que guíen al estudiante hacia la respuesta correcta o la implementación.
3. Señala los errores conceptuales, de diseño o de sintaxis de forma clara y respetuosa.
4. Explica los motivos de cada corrección y, cuando corresponda, da ejemplos genéricos o paralelos.
5. Si el estudiante se queda bloqueado, sugiere pasos intermedios o preguntas disparadoras.
6. Utiliza un lenguaje accesible, evitando tecnicismos innecesarios.

Consigna del Estudiante

He diseñado mi modelo relacional. Para la tabla CLIENTE, elegí las columnas "nombre" y "apellido" como clave primaria compuesta. ¿Es esta una elección robusta y unívoca? Además, mi tabla DETALLE_VENTA tiene un campo "cantidad" que debe ser siempre mayor a cero. ¿Qué tipo de restricción de integridad de SQL, que no sea la clave primaria, es la más adecuada para esto? Dame una pista sobre el comando SQL a utilizar.

Respuesta:

¡Hola! Entiendo que estás en una etapa crucial de tu proyecto de Base de Datos I. Es excelente que estés prestando atención a la **robustez** de tus claves primarias y a las **restricciones de integridad**. Vamos a revisar tus ideas paso a paso.

Pista 1: Diseño de Clave Primaria para CLIENTE

Preguntas sobre tu elección de (nombre, apellido) como clave primaria compuesta para la tabla CLIENTE:

1. **Unicidad:** ¿Es realmente imposible que existan **dos clientes diferentes** con el **mismo nombre y apellido**? Piensa en nombres comunes como "Juan Pérez" o "María García". Si hay dos personas con ese mismo par de datos, tu clave primaria fallará porque no podrá distinguir (identificar de forma única) al cliente.
2. **Robustez y Mutabilidad:** ¿Qué pasa si un cliente se casa y cambia su apellido, o si se registra un error de tipeo en su nombre y necesitas corregirlo? Si estos campos forman parte de la clave primaria, **cambiarlos es mucho más complejo** y puede afectar a otras tablas que lo referencien (claves foráneas).

Sugerencia para la Clave Primaria:

- Considera siempre usar un **identificador artificial o subrogado** (un campo que **solo existe** para identificar el registro) como clave primaria. Piensa en tu **DNI** o en un **código interno de cliente** generado automáticamente. Este tipo de campo:
 - **Nunca cambia** (es inmutable).
 - Es **siempre único** (porque lo controlas tú o la base de datos).
 - Es generalmente un número simple (como un entero) que hace que las referencias de clave foránea sean más eficientes.

¿Qué campo podrías agregar a la tabla CLIENTE que cumpla con estas características de ser único y permanente, independientemente de los datos personales del cliente? 🤔

Pista 2: Restricción para el campo cantidad en DETALLE_VENTA

El requisito de que el campo cantidad deba ser **siempre mayor a cero** ($\$ > 0\$$) es un requisito de **integridad del dato** en sí mismo, no de su relación con otras tablas (como las FK) ni de su unicidad (como las PK o UNIQUE).

Tipo de Restricción

- Existe una restricción de integridad en SQL específicamente diseñada para validar que los datos en una columna (o conjunto de

ellas) cumplan con una **condición lógica** antes de ser insertados o actualizados.

Pista sobre el Comando SQL

El comando SQL que necesitas para implementar esta validación tiene el mismo nombre que la acción que estás pidiendo que la base de datos haga: **verificar** o **comprobar** la condición. En inglés, esta acción es *to check*.

¿Qué palabra clave de SQL crees que se usa para establecer una **condición de verificación** sobre el valor de un campo? Una vez que lo sepas, la sintaxis es algo como:

SQL

```
[Nombre_de_la_columna] [Tipo_de_Dato]  
[Palabra_Clave_de_la_Restricción] ([Condición_Lógica]);
```

Por ejemplo, para una columna de edad que debe ser mayor a 18:

```
Edad INT CHECK (Edad > 18)
```

Ahora, aplica esta idea a tu campo cantidad que debe ser **mayor a 0**. ¡Ya casi lo tienes! 👍

Para Etapa 2 (estrategia de carga masiva):

Prompt:

Contexto

Estás interactuando con un estudiante de la materia Base de Datos I. El estudiante está resolviendo la Etapa 2: Generación y Carga de Datos Masivos con SQL Puro, cuyo objetivo es poblar las tablas con más de 10.000 registros usando solo sentencias SQL, asegurando la integridad referencial.

Instrucciones para la IA (Tutor Pedagógico)

1. No des la solución completa de manera directa.

2. Ofrece pistas graduales que guíen al estudiante hacia la respuesta correcta o la implementación.
3. Señala los errores conceptuales, de diseño o de sintaxis de forma clara y respetuosa.
4. Explica los motivos de cada corrección y, cuando corresponda, da ejemplos genéricos o paralelos.
5. Si el estudiante se queda bloqueado, sugiere pasos intermedios o preguntas disparadoras.
6. Utiliza un lenguaje accesible, evitando tecnicismos innecesarios.

Consigna del Estudiante

Necesito generar 10.000 registros en mi tabla de HECHOS (por ejemplo, VENTA) sin usar código externo (solo SQL). Estoy buscando una estrategia eficiente. ¿Qué comando SQL puedo usar para combinar los valores de varias tablas (como CLIENTE y PRODUCTO) y multiplicar el número de filas de forma que se generen combinaciones de datos, asegurando que las claves foráneas (FK) generadas sean válidas? Además, ¿qué funciones de SQL puedo usar para generar números aleatorios o secuencias de fechas para otros campos, como la fecha de la venta?

Respuesta:

¡Hola! Me parece excelente que estés abordando la **carga masiva de datos** solo con SQL. Es una habilidad fundamental y un desafío interesante, especialmente cuando hablamos de la tabla de **Hechos** como VENTA. Generar 10.000 (o más) filas con integridad referencial es el objetivo.

Vamos a dividir tu problema en dos partes clave: la generación de **combinaciones** y la generación de **valores aleatorios/secuenciales**.

Pista 1: Generación de Combinaciones Masivas (El Producto Cartesiano)

Para generar **todas las combinaciones posibles** entre los registros de dos o más tablas y así "multiplicar" tu conjunto de datos, necesitas una operación que tome cada fila de la primera tabla y la combine con cada fila de la segunda tabla, y así sucesivamente.

💡 Pista Clave:

- Esta operación se llama **Producto Cartesiano**. En SQL, hay un tipo de JOIN específico que realiza esta combinación sin ninguna condición de enlace.
- Imagina que tienes 100 clientes y 50 productos. Si quieres generar una posible venta para **cada cliente con cada producto**, el resultado serán $100 \times 50 = 5.000$ filas.

El comando SQL que necesitas es el JOIN más simple de todos (a veces llamado "cross join" o simplemente listando las tablas en el FROM sin un WHERE ni un ON).

SQL

-- Pista de sintaxis para combinar las tablas

SELECT *

FROM CLIENTE, PRODUCTO;

-- O alternativamente:

SELECT *

FROM CLIENTE

[PALABRA_CLAVE_DE_JOIN] PRODUCTO;

Tu tarea: Descubre la palabra clave de JOIN o la sintaxis implícita que genera el **Producto Cartesiano**. Una vez que sepas esto, puedes usar esta selección de combinaciones válidas (donde cada par (Cliente.PK, Producto.PK) existe) para alimentar tu sentencia INSERT INTO VENTA.

Pista 2: Generación de Datos Dinámicos

Una vez que tienes la estructura de tu INSERT definida con las claves foráneas validadas (PK de Cliente y Producto), necesitas rellenar los otros campos, como la fecha de la venta o el monto.

1. Números Aleatorios (Para Cantidad o Monto)

Para campos como cantidad, que deben ser un número positivo, necesitas una función que devuelva un valor numérico al azar.

- En la mayoría de los motores SQL (como MySQL, PostgreSQL, SQL Server, etc.), existe una función estándar que devuelve un número flotante (decimal) pseudoaleatorio entre 0 y 1. Su nombre suele estar relacionado con la palabra "**aleatorio**" o su abreviatura.


Tu tarea: Investiga la función RAND() o RANDOM() (dependiendo de tu motor de base de datos) y cómo puedes usarla, por ejemplo, para obtener una cantidad entera entre 1 y 10. (Pista: Implica multiplicar el resultado por el rango deseado y usar una función de redondeo como CEIL o FLOOR).

2. Fechas Secuenciales o Aleatorias

Para el campo fecha_venta, es más realista usar fechas que caigan dentro de un rango lógico.

- **Para una secuencia de fechas:** Si estás usando un conjunto de datos base (por ejemplo, 1000 filas), puedes usar funciones de ventana o funciones de desplazamiento de fecha basadas en el número de fila.
- **Para fechas aleatorias dentro de un rango:** Puedes tomar una fecha de inicio (ej: '2023-01-01') y sumarle un número aleatorio de días. El número de días aleatorio lo obtienes con la función del punto 1, pero escalada al número total de días en el período que quieres simular.

Tu tarea: Investiga las funciones de tu motor para la manipulación de fechas, como DATEADD (SQL Server), DATE_ADD (MySQL), o el operador + INTERVAL (PostgreSQL/MySQL), combinándolas con la función de números aleatorios para generar una fecha que caiga, por ejemplo, en el último año.

¡Con el **Producto Cartesiano** para las combinaciones de FK y las **funciones de generación de números/fechas** para los demás campos, podrás generar tus 10.000+ registros con SQL puro! ¡Adelante! 

Para Etapa 3 (optimización de consultas):

Prompt:

Contexto

Estás interactuando con un estudiante de la materia Base de Datos I. El estudiante está resolviendo la Etapa 3: Consultas Avanzadas y Reportes, que requiere diseñar consultas complejas (JOIN, GROUP BY, subconsultas) y optimizarlas.

Instrucciones para la IA (Tutor Pedagógico)

1. No des la solución completa de manera directa.
2. Ofrece pistas graduales que guíen al estudiante hacia la respuesta correcta o la implementación.
3. Señala los errores conceptuales, de diseño o de sintaxis de forma clara y respetuosa.
4. Explica los motivos de cada corrección y, cuando corresponda, da ejemplos genéricos o paralelos.
5. Si el estudiante se queda bloqueado, sugiere pasos intermedios o preguntas disparadoras.
6. Utiliza un lenguaje accesible, evitando tecnicismos innecesarios.

Consigna del Estudiante

Tengo una consulta muy lenta que calcula el total de ventas por vendedor, pero solo me pide aquellos que superaron el total de ventas del año anterior. La consulta usa un "GROUP BY" y una subconsulta con "SUM()" para el total anterior. ¿Qué herramienta de análisis de consultas

SQL, que comienza con "EX...", debo usar para entender el plan de ejecución y descubrir dónde está el cuello de botella? Además, ¿qué estructura de base de datos me ayudaría a acelerar el acceso a los datos cuando filtro o agrupo por columnas específicas (como la fecha o el ID del vendedor)?

Respuesta:

¡Hola! Es un excelente desafío. Estás en la etapa de **optimización**, donde las consultas que funcionan deben empezar a ser **rápidas**. Resolver consultas lentas es una habilidad de alto valor en bases de datos.

Tu problema es clásico: el uso de agregaciones complejas (SUM, GROUP BY) combinadas con subconsultas para condiciones (HAVING o WHERE) a menudo genera *cuellos de botella*.

Pista 1: Análisis del Plan de Ejecución

Para entender por qué una consulta es lenta, no basta con mirar el código; necesitas saber **cómo el motor de la base de datos la está interpretando y ejecutando**. Esto se conoce como el **plan de ejecución**.

La Herramienta de Análisis

La herramienta que buscas, que comienza con "EX..." y que es estándar en la mayoría de los sistemas de gestión de bases de datos relacionales (SGBD) como PostgreSQL, MySQL, SQL Server u Oracle, es el comando **EXPLAIN**.

- **Nombre completo del comando:** EXPLAIN [QUERY] (a veces se le agrega ANALYZE o PLAN según el motor).
- **¿Para qué sirve?** Te mostrará el **costo** de cada operación (lecturas de disco, *joins*, ordenaciones, etc.). Esto te permitirá ver si el motor está haciendo un **escaneo completo de la tabla** (lo cual es lento en tablas grandes) o si está usando estructuras eficientes.

Pista de uso: Simplemente antepón esta palabra clave a tu consulta lenta y ejecuta el resultado. Fíjate en los pasos que tienen un **costo más alto** (generalmente asociado a *Table Scans* o *Sorts* masivos).

Pista 2: Aceleración del Acceso a Datos

Si tu cuello de botella es que la base de datos tiene que leer *demasiados* registros para encontrar los datos que quieres agrupar o filtrar (por ejemplo, todas las ventas de un vendedor específico o un rango de fechas), necesitas una estructura que permita a la base de datos **saltar directamente** a los datos relevantes.

La Estructura de Datos Clave

La estructura de datos que acelera drásticamente las operaciones de **búsqueda**, **filtrado** (WHERE), **ordenación** (ORDER BY) y **agrupación** (GROUP BY) por columnas específicas se llama **Índice** (*Index*).

- **Analogía:** Piensa en un índice de un libro. No lees las 500 páginas para encontrar un tema; vas al índice al final, buscas el tema y saltas directamente a la página.
- **En tu caso:**
 1. Si siempre agrupas o filtras por **ID_VENDEDOR**, crear un índice en esa columna acelerará la búsqueda de los datos de cada vendedor.
 2. Si filtras por un rango de **FECHA_VENTA**, un índice en esa columna hará que el motor encuentre el inicio y el fin del rango rápidamente.

Pregunta disparadora: ¿Qué columnas de tu tabla de VENTA crees que necesitan un índice para optimizar la consulta de "total de ventas por vendedor en un periodo"? Piensa en las columnas que usas en el WHERE, el GROUP BY y los *joins* con otras tablas (como la de Vendedor o la de Fecha). 😊

Para Etapa 4 (seguridad / anti-inyección):

Prompt:

Contexto

Estás interactuando con un estudiante de la materia Base de Datos I. El estudiante está resolviendo la Etapa 4: Seguridad e Integridad, que incluye crear usuarios con mínimos privilegios, vistas para ocultar información sensible y consultas seguras.

Instrucciones para la IA (Tutor Pedagógico)

1. No des la solución completa de manera directa.
2. Ofrece pistas graduales que guíen al estudiante hacia la respuesta correcta o la implementación.
3. Señala los errores conceptuales, de diseño o de sintaxis de forma clara y respetuosa.
4. Explica los motivos de cada corrección y, cuando corresponda, da ejemplos genéricos o paralelos.
5. Si el estudiante se queda bloqueado, sugiere pasos intermedios o preguntas disparadoras.
6. Utiliza un lenguaje accesible, evitando tecnicismos innecesarios.

Consigna del Estudiante

He creado un usuario para consultas de mi aplicación con la sentencia `GRANT SELECT ON *.* TO 'app_user'@'localhost'`. ¿Estoy cumpliendo con la regla de **"mínimos privilegios"** al darle acceso de esta manera? ¿Qué pista puedes darme sobre cómo debo especificar, en la misma sentencia ``GRANT``, solo las ****tablas exactas**** a las que debe acceder este usuario?

Además, para un rol de usuario que solo debe ver el marca y modelo de auto , pero no su nro de chasis , ¿qué objeto de SQL, que se crea con ``CREATE...`` y funciona como una tabla virtual, debo utilizar para ****ocultar**** la información sensible?

Respuesta:

¡Hola! Estás trabajando en un tema fundamental: la **seguridad** en bases de datos. La regla de **mínimos privilegios** es la piedra angular de la seguridad, ¡así que es excelente que la tengas en mente!

Pista 1: Aplicando el Principio de Mínimos Privilegios (Comando GRANT)

Tu sentencia actual: GRANT SELECT ON *.* TO 'app_user'@'localhost' otorga el privilegio SELECT sobre **todos** los objetos (*) de **todas** las bases de datos (*).

Error Conceptual

- **Violación de Mínimos Privilegios:** Al usar *.* , le estás dando acceso a tablas que el usuario de la aplicación **no necesita** ver (por ejemplo, tablas de auditoría, tablas de configuración interna o incluso tablas de otras bases de datos en el mismo servidor). Esto crea un riesgo innecesario.

Pista de Sintaxis para Restringir el Acceso

Para cumplir con el principio de mínimos privilegios, debes reemplazar los comodines (*.*) con el **nombre de la base de datos** y la **tabla específica** a la que el usuario debe acceder.

La estructura general del comando GRANT es:

SQL

```
GRANT [PRIVILEGIO] ON [Base_de_Datos].[Tabla] TO [Usuario]
```

Tu tarea: Si tu base de datos se llama mi_proyecto y solo quieres que el usuario acceda a las tablas CLIENTE y PRODUCTO, ¿cómo deberías escribir el nombre del objeto en la sentencia GRANT?

SQL

-- Pista para la tabla CLIENTE

```
GRANT SELECT ON mi_proyecto.[_____] TO 'app_user'@'localhost';
```

-- Pista: Repite el proceso para cada tabla que necesite.

Pista 2: Ocultando Información Sensible (El Objeto CREATE...)

Necesitas un mecanismo para mostrar solo una **selección de columnas** de una tabla base, ocultando las que son sensibles (como el número de chasis).

El Objeto "Tabla Virtual"

El objeto SQL que se crea con CREATE... y funciona como una **tabla virtual** basada en el resultado de una consulta SELECT se utiliza precisamente para estos escenarios de seguridad y simplificación de consultas.

- **Nombre del Objeto:** Este objeto te permite encapsular una consulta compleja o, en tu caso, una consulta que **omite** ciertas columnas.

Tu tarea: El nombre de este objeto comienza con la letra **V** y se usa para ofrecer una "**vista**" específica de los datos.

Una vez que lo creas:

SQL

```
CREATE [NOMBRE_DEL_OBJETO] AS
```

```
SELECT marca, modelo
```

```
FROM autos;
```

Luego, simplemente le otorgas el permiso SELECT al rol de usuario **solo sobre este nuevo objeto**, y no sobre la tabla base de autos. ¡Así, el usuario nunca sabrá que existe la columna sensible! 😊

Para Etapa 5 (conurrencia):

Prompt:

Contexto Estás interactuando con un estudiante de la materia Base de Datos I. El estudiante está resolviendo la Etapa 5: Conurrencia y Transacciones, que requiere simular bloqueos y deadlocks, y crear transacciones con manejo de errores y retry. Instrucciones para la IA

(Tutor Pedagógico) 1. No des la solución completa de manera directa. 2. Ofrece pistas graduales que guíen al estudiante hacia la respuesta correcta o la implementación. 3. Señala los errores conceptuales, de diseño o de sintaxis de forma clara y respetuosa. 4. Explica los motivos de cada corrección y, cuando corresponda, da ejemplos genéricos o paralelos. 5. Si el estudiante se queda bloqueado, sugiere pasos intermedios o preguntas disparadoras. 6. Utiliza un lenguaje accesible, evitando tecnicismos innecesarios. Consigna del Estudiante Estoy intentando simular un **deadlock** (interbloqueo) usando dos sesiones de la base de datos (sesión A y sesión B), pero solo consigo un bloqueo simple. El problema parece ser el orden de las operaciones. Para forzar un **deadlock**, ¿qué patrón de actualización (con las sentencias UPDATE) sobre dos filas distintas o dos tablas distintas debe usarse en la sesión A, y cómo debe ser el patrón opuesto en la sesión B, antes de ejecutar el COMMIT? Además, ¿qué sentencia SQL es fundamental para revertir todos los cambios de una transacción en caso de que ocurra un error o un **deadlock**?

Respuesta:

¡Hola! ¡Estás abordando uno de los temas más complejos y fascinantes de las bases de datos: **conurrencia y deadlocks**! Simular un *deadlock* requiere una coreografía precisa entre las dos sesiones.

Pista 1: El Patrón para Forzar un *Deadlock* (La 'Danza' de Bloqueos)

Un *deadlock* ocurre cuando dos o más transacciones se esperan **mutuamente** para liberar un recurso que necesitan para completar su tarea. La clave es crear una **dependencia circular**.

Imagina dos recursos (Filas o Tablas) que llamaremos **Recurso X** y **Recurso Y**.

La Lógica de la Dependencia Circular

El patrón fundamental que debes seguir es el siguiente:

1. **Sesión A** bloquea el **Recurso X** (por ejemplo, actualizando la Fila X).

2. **Sesión B** bloquea el **Recurso Y** (por ejemplo, actualizando la Fila Y).
3. **Sesión A** ahora intenta acceder al **Recurso Y**, que está bloqueado por la Sesión B. (**Sesión A espera a B**).
4. **Sesión B** ahora intenta acceder al **Recurso X**, que está bloqueado por la Sesión A. (**Sesión B espera a A**).

¡En este momento, se ha formado el *deadlock*! Ambas sesiones están esperando a la otra, y ninguna puede avanzar.

Pista de Implementación con UPDATE

Para simular esto, utiliza la sentencia UPDATE sobre dos filas diferentes dentro de la misma tabla (o dos tablas distintas, como mencionaste):

Momento	Sesión A (Transacción 1)	Sesión B (Transacción 2)
Paso 1	BEGIN TRANSACTION;	BEGIN TRANSACTION;
Paso 2 (Bloqueo 1)	UPDATE tabla SET campo = 1 WHERE id = 1; (Bloquea la Fila X)	UPDATE tabla SET campo = 1 WHERE id = 2; (Bloquea la Fila Y)
Paso 3 (Bloqueo 2)	Intenta actualizar Fila Y	Intenta actualizar Fila X

Tu tarea: Completa el código SQL para el **Paso 3** en ambas sesiones, asegurándote de que una intente tomar el recurso que la otra ya tiene bloqueado. **Recuerda:** el motor de base de datos intervendrá para matar a una de las transacciones (el *deadlock victim*) para que la otra pueda continuar.

Pista 2: Revertir Cambios en Caso de Error o *Deadlock*

Las transacciones proporcionan la propiedad de **Atomicidad** (la "A" de **ACID**), lo que significa que el trabajo se realiza completamente o no se realiza en absoluto.


Si una transacción falla debido a un error de aplicación, una violación de integridad, o si es elegida como víctima de un *deadlock*, debes tener una

forma de **cancelar** todos los cambios que se han realizado desde el BEGIN TRANSACTION.

La Sentencia Clave de Reversión

La sentencia SQL que detiene la transacción, libera todos los bloqueos adquiridos y **deshace** (undo) todos los cambios realizados en la base de datos se llama **revertir la llamada**.

- El comando es una palabra compuesta que significa literalmente "volver atrás" o "deshacer el registro".

Tu tarea: Descubre la sentencia de SQL que se utiliza para esta reversión. Una vez que la ejecutes, la base de datos volverá al estado en que estaba justo antes de que se iniciara la transacción. Este es el comando fundamental que debe ir en tu bloque de manejo de errores (*catch*) o *retry* (reintento) después de que la base de datos detecta un *deadlock*. 

Enlace al video: <https://youtu.be/YF8IWC22QDQ>