

# Trabajo Práctico Integrador

## Alumnos

Valentin Piñeyro, Pérez Lucio, Matias Pérez y Danilo Peirano

## Tecnicatura Universitaria en Programación

## Universidad Tecnológica Nacional.

## Base de Datos I

### Docente Titular

Gustavo Sturtz

### Docente Tutor

Miguel Tola

### Link Video

[https://drive.google.com/drive/folders/1KU0t0-WkpGT8GpwaKd5leEyf0ahqHmq1m?usp=drive\\_link](https://drive.google.com/drive/folders/1KU0t0-WkpGT8GpwaKd5leEyf0ahqHmq1m?usp=drive_link)

23 de octubre de 2025

## ETAPA 1: Modelado y constraints

**Dominio:** se modela un sistema de ventas que gestiona clientes y sus perfiles, además registra las ventas realizadas.

### Reglas principales:

- **clientes.dni** es único y tiene 8 dígitos.
- **ventas.total** debe ser mayor o igual a 0.
- **ventas.estado** solo puede ser: PENDIENTE, PAGADA o CANCELADA.
- **perfiles\_clientes** tiene la misma PK que **clientes** (relación 1:1) y su campo **email** es único.

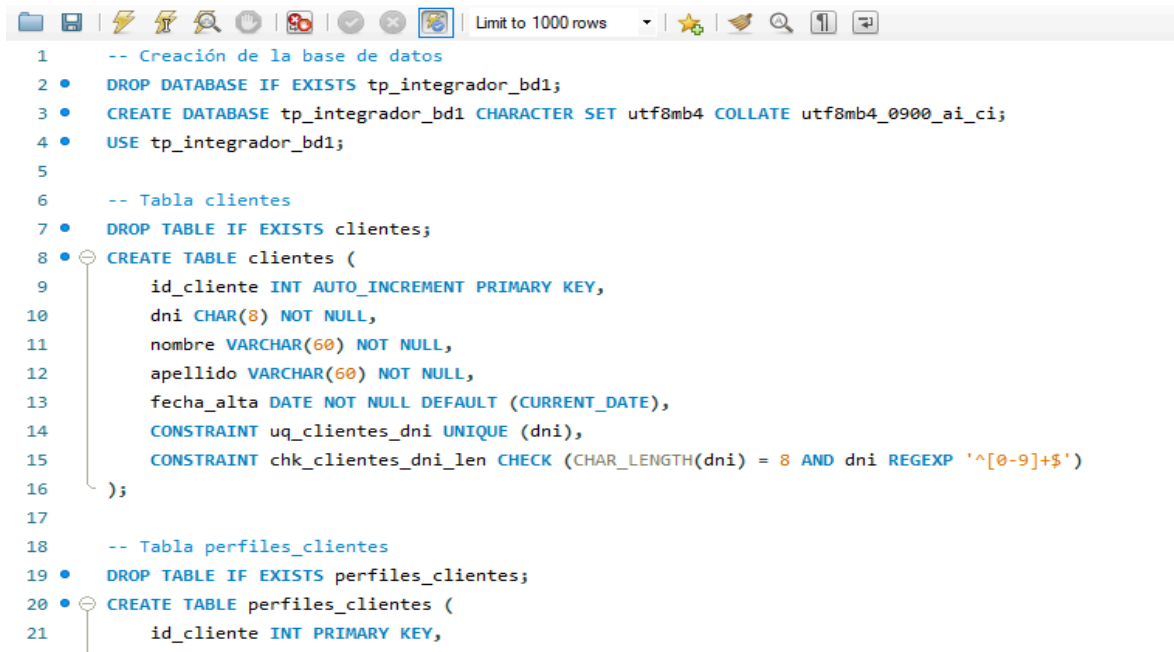
### Decisiones de diseño:

Se implementan constraints de integridad (PK, FK, UNIQUE, CHECK) para garantizar la consistencia.

Se usa **ON DELETE CASCADE** entre **clientes** y **perfiles\_clientes** para eliminar perfiles cuando se borra un cliente.

### Parte práctica (scripts SQL)

Se realizaron dos inserciones válidas (Ana Sosa y Luis Videla) y dos erróneas (DNI duplicado y total negativo) para comprobar los constraints.



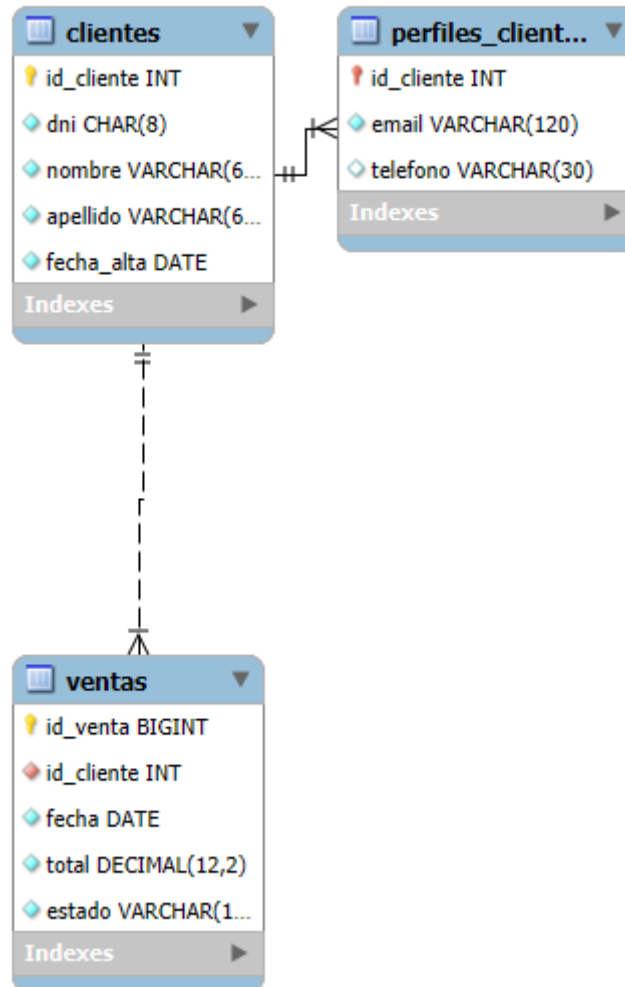
```
1  -- Creación de la base de datos
2  • DROP DATABASE IF EXISTS tp_integrador_bd1;
3  • CREATE DATABASE tp_integrador_bd1 CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci;
4  • USE tp_integrador_bd1;
5
6  -- Tabla clientes
7  • DROP TABLE IF EXISTS clientes;
8  • CREATE TABLE clientes (
9      id_cliente INT AUTO_INCREMENT PRIMARY KEY,
10     dni CHAR(8) NOT NULL,
11     nombre VARCHAR(60) NOT NULL,
12     apellido VARCHAR(60) NOT NULL,
13     fecha_alta DATE NOT NULL DEFAULT (CURRENT_DATE),
14     CONSTRAINT uq_clientes_dni UNIQUE (dni),
15     CONSTRAINT chk_clientes_dni_len CHECK (CHAR_LENGTH(dni) = 8 AND dni REGEXP '^[0-9]+$')
16 );
17
18 -- Tabla perfiles_clientes
19 • DROP TABLE IF EXISTS perfiles_clientes;
20 • CREATE TABLE perfiles_clientes (
21     id_cliente INT PRIMARY KEY,
```

```

1  USE tp_integrador_bd1;
2
3  -- Inserciones CORRECTAS (2 casos)
4  -- Caso A
5  • INSERT INTO clientes (dni, nombre, apellido) VALUES ('12345678','Ana','Pérez');
6  • INSERT INTO perfiles_clientes (id_cliente, email, telefono)
7  SELECT id_cliente, 'ana@example.com', '351-1111111' FROM clientes WHERE dni='12345678';
8  • INSERT INTO ventas (id_cliente, fecha, total, estado)
9  SELECT id_cliente, '2025-09-01', 25000.00, 'PAGADA' FROM clientes WHERE dni='12345678';
10
11 -- Caso B
12 • INSERT INTO clientes (dni, nombre, apellido) VALUES ('23456789','Luis','Gómez');
13 • INSERT INTO perfiles_clientes (id_cliente, email)
14 SELECT id_cliente, 'luis@example.com' FROM clientes WHERE dni='23456789';
15 • INSERT INTO ventas (id_cliente, fecha, total, estado)
16 SELECT id_cliente, '2025-09-02', 0.00, 'PENDIENTE' FROM clientes WHERE dni='23456789';
17
18 -- Inserciones ERRÓNEAS (2 tipos distintos)
19 -- Error tipo UNIQUE (dni duplicado)
20 • INSERT INTO clientes (dni, nombre, apellido) VALUES ('12345678','Ana Duplicada','X');
21
22 -- Error tipo CHECK (total negativo)
23 • INSERT INTO ventas (id_cliente, fecha, total, estado)
24 SELECT id_cliente, '2025-09-03', -10.00, 'PAGADA' FROM clientes WHERE dni='12345678';

```

#	Time	Action	Message	Duration / Fetch
1	01:04:07	DROP DATABASE IF EXISTS tp_integrador_bd1	0 row(s) affected, 1 warning(s): 1008 Can't drop database 'tp_integrador_bd1': database doesn't exist	0.000 sec
2	01:04:07	CREATE DATABASE tp_integrador_bd1 CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci	1 row(s) affected	0.000 sec
3	01:04:07	USE tp_integrador_bd1	0 row(s) affected	0.000 sec
4	01:04:07	DROP TABLE IF EXISTS clientes	0 row(s) affected, 1 warning(s): 1051 Unknown table 'tp_integrador_bd1.clientes'	0.000 sec
5	01:04:07	CREATE TABLE clientes ( id_cliente INT AUTO_INCREMENT PRIMARY KEY, dni CHAR(8) NOT NULL, nombre ...	0 row(s) affected	0.015 sec
6	01:04:07	DROP TABLE IF EXISTS perfiles_clientes	0 row(s) affected, 1 warning(s): 1051 Unknown table 'tp_integrador_bd1.perfiles_clientes'	0.016 sec
7	01:04:07	CREATE TABLE perfiles_clientes ( id_cliente INT PRIMARY KEY, email VARCHAR(120) NOT NULL, telefono VAR...	0 row(s) affected	0.016 sec
8	01:04:07	DROP TABLE IF EXISTS ventas	0 row(s) affected, 1 warning(s): 1051 Unknown table 'tp_integrador_bd1.ventas'	0.000 sec
9	01:04:07	CREATE TABLE ventas ( id_venta BIGINT AUTO_INCREMENT PRIMARY KEY, id_cliente INT NOT NULL, fecha...	0 row(s) affected	0.015 sec
10	01:14:32	USE tp_integrador_bd1	0 row(s) affected	0.000 sec
11	01:14:32	INSERT INTO clientes (dni, nombre, apellido) VALUES (12345678,'Ana','Pérez')	1 row(s) affected	0.000 sec
12	01:14:32	INSERT INTO perfiles_clientes (id_cliente, email, telefono) SELECT id_cliente, 'ana@example.com', '351-1111111' FROM ...	1 row(s) affected Records: 1 Duplicates: 0 Warnings: 0	0.000 sec
13	01:14:32	INSERT INTO ventas (id_cliente, fecha, total, estado) SELECT id_cliente, '2025-09-01', 25000.00, 'PAGADA' FROM client...	1 row(s) affected Records: 1 Duplicates: 0 Warnings: 0	0.016 sec
14	01:14:32	INSERT INTO clientes (dni, nombre, apellido) VALUES (23456789,'Luis','Gómez')	1 row(s) affected	0.000 sec
15	01:14:32	INSERT INTO perfiles_clientes (id_cliente, email) SELECT id_cliente, 'luis@example.com' FROM clientes WHERE dni=234...	1 row(s) affected Records: 1 Duplicates: 0 Warnings: 0	0.000 sec
16	01:14:32	INSERT INTO ventas (id_cliente, fecha, total, estado) SELECT id_cliente, '2025-09-02', 0.00, 'PENDIENTE' FROM cliente...	1 row(s) affected Records: 1 Duplicates: 0 Warnings: 0	0.000 sec
17	01:14:32	INSERT INTO clientes (dni, nombre, apellido) VALUES (12345678,'Ana Duplicada','X')	Error Code: 1062. Duplicate entry '12345678' for key 'clientes.uq_clientes_dni'	0.000 sec



## Uso de IA

### Objetivo de la consulta:

Solicite ayuda para iniciar la Etapa 1 desde cero, definir un modelo mínimo viable y obtener los scripts SQL de creación y pruebas.

### Qué generó la IA:

- Propuesta de dominio simple (clientes ↔ perfiles ↔ ventas).
- Scripts idempotentes con PK, FK, UNIQUE y CHECK.
- Ejemplos de inserciones válidas y erróneas para evidenciar constraints.
- Guía de entrega y estructura de informe.

### Decisiones del equipo:

- Mantener MySQL 8 como motor.
- Usar relación 1:1 entre clientes y perfiles y 1:N con ventas.
- Definir estados de venta: PENDIENTE, PAGADA, CANCELADA.

**Reflexión propia:**

Comprendimos cómo las restricciones de integridad garantizan la coherencia de los datos y cómo documentar los errores ayuda a validar el modelo. El uso de IA nos permitió ahorrar tiempo en estructura y centrarnos en entender las reglas de negocio.

## ***Etapa 2 - Generación y carga de datos masivos con SQL puro.***

**Objetivo:**

- Generar 10.000 clientes y sus perfiles (2 tablas).
- Generar 50.000 ventas aleatorias vinculadas a esos clientes.
- Medir tiempos antes y después de crear un índice (ejemplo de consulta típica).
- Realizar verificaciones y conteos.

### ***Descripción:***

El mecanismo implementado se basa en el uso de tablas semilla para generar datos de manera masiva y controlada, asegurando la consistencia referencial entre las distintas entidades del modelo. La tabla semilla principal (**t\_nums\_10000**) se construye a partir de combinaciones cruzadas de dígitos, generando un conjunto numérico del 1 al 10.000 que sirve como base para crear registros simulados en las tablas maestras.

A partir de esta tabla auxiliar, se insertan datos en la tabla **clientes**, que funciona como entidad principal. Cada cliente se identifica por un **id\_cliente** único y contiene información básica como DNI, nombre, apellido y fecha de alta. Posteriormente, se genera la tabla **perfiles\_clientes**, que mantiene una relación **uno a uno (1:1)** con la tabla de clientes. Esta relación se garantiza utilizando el mismo **id\_cliente** como clave foránea, de modo que cada perfil corresponde exactamente a un cliente y no puede existir sin él.

La tabla **ventas** representa la entidad transaccional del sistema y está vinculada a la tabla de clientes mediante la clave foránea **id\_cliente**, estableciendo una relación **uno a muchos (1:N)**, ya que cada cliente puede registrar múltiples ventas. La integridad referencial se mantiene mediante las restricciones de clave foránea y la inserción controlada de datos, que asegura que todos los identificadores utilizados existan previamente en las tablas maestras.

Este enfoque permite simular un entorno de base de datos relacional realista, donde las **tablas semilla** facilitan la creación de grandes volúmenes de información coherente, preservando las **cardinalidades e integridad referencial**, fundamentales para garantizar la consistencia del modelo de datos.

### ***Capturas:***

```
1 • USE tp_integrador_bd1;
2
3 -- Limpieza para regenerar datos
4 SET FOREIGN_KEY_CHECKS = 0;
5 TRUNCATE TABLE ventas;
6 TRUNCATE TABLE perfiles_clientes;
7 TRUNCATE TABLE clientes;
8 SET FOREIGN_KEY_CHECKS = 1;
9
10 -- Tabla auxiliar para números
11 • DROP TEMPORARY TABLE IF EXISTS t_dig;
12 • CREATE TEMPORARY TABLE t_dig (d TINYINT UNSIGNED NOT NULL);
13 • INSERT INTO t_dig (d) VALUES (0),(1),(2),(3),(4),(5),(6),(7),(8),(9);
14 • DROP TEMPORARY TABLE IF EXISTS t_nums_10000;
15 • CREATE TEMPORARY TABLE t_nums_10000 (n INT PRIMARY KEY);
16
17 • DROP TABLE IF EXISTS t_dig_a;
18 • DROP TABLE IF EXISTS t_dig_b;
19 • DROP TABLE IF EXISTS t_dig_c;
20 • DROP TABLE IF EXISTS t_dig_d;
21
22 • CREATE TABLE t_dig_a AS SELECT * FROM t_dig;
23 • CREATE TABLE t_dig_b AS SELECT * FROM t_dig;
24 • CREATE TABLE t_dig_c AS SELECT * FROM t_dig;
25 • CREATE TABLE t_dig_d AS SELECT * FROM t_dig;
26
27 • INSERT INTO t_nums_10000 (n)
28   SELECT (a.d*1000 + b.d*100 + c.d*10 + d.d) + 1 AS n
29   FROM t_dig_a a
30   CROSS JOIN t_dig_b b
31   CROSS JOIN t_dig_c c
32   CROSS JOIN t_dig_d d;
33
34 • INSERT INTO clientes (dni, nombre, apellido, fecha_alta)
35   SELECT
36     LPAD(10000000 + n, 8, '0') AS dni, -- 8 dígitos
37     CONCAT('Cliente ', LPAD(n,5,'0')) AS nombre,
38     CONCAT('Apellido ', LPAD(n,5,'0')) AS apellido,
39     DATE_ADD('2024-01-01', INTERVAL FLOOR(RAND()*365) DAY) AS fecha_alta
40   FROM t_nums_10000;
```



```

40
41 -- 3) Generar 10.000 PERFILES (1:1 con clientes)
42 -- Emparejamos por id_cliente en orden de inserción
43 • INSERT INTO perfiles_clientes (id_cliente, email, telefono)
44 SELECT
45     c.id_cliente,
46     CONCAT('cliente', LPAD(@rownum:=@rownum+1,5,'0'), '@example.com') AS email,
47     CONCAT('+54 351 ', LPAD(FLOOR(RAND()*10000000),7,'0')) AS telefono
48 FROM (SELECT @rownum:=0) vars
49 JOIN (SELECT id_cliente FROM clientes ORDER BY id_cliente ASC LIMIT 10000) c;
50 -- 4) Generar 50.000 VENTAS
51 -- Usamos otra tabla temporal de 50k filas (5 * 10k)
52 • DROP TEMPORARY TABLE IF EXISTS t_nums_50000;
53 • CREATE TEMPORARY TABLE t_nums_50000 (n INT PRIMARY KEY);
54
55 • DROP TABLE IF EXISTS t_nums_10000_a;
56 • DROP TABLE IF EXISTS t_nums_10000_b;
57 • DROP TABLE IF EXISTS t_nums_10000_c;
58 • DROP TABLE IF EXISTS t_nums_10000_d;
59 • DROP TABLE IF EXISTS t_nums_10000_e;
60
61 • CREATE TABLE t_nums_10000_a AS SELECT * FROM t_nums_10000;
62 • CREATE TABLE t_nums_10000_b AS SELECT * FROM t_nums_10000;
63 • CREATE TABLE t_nums_10000_c AS SELECT * FROM t_nums_10000;
64 • CREATE TABLE t_nums_10000_d AS SELECT * FROM t_nums_10000;
65 • CREATE TABLE t_nums_10000_e AS SELECT * FROM t_nums_10000;
--

```

### Salida:

148	07:42:02	USE db1	0 row(s) affected	0.000 sec
149	07:42:02	SET FOREIGN_KEY_CHECKS = 0	0 row(s) affected	0.000 sec
150	07:42:02	TRUNCATE TABLE ventas	0 row(s) affected	0.031 sec
151	07:42:02	TRUNCATE TABLE perfiles_clientes	0 row(s) affected	0.031 sec
152	07:42:02	TRUNCATE TABLE clientes	0 row(s) affected	0.031 sec
153	07:42:02	SET FOREIGN_KEY_CHECKS = 1	0 row(s) affected	0.000 sec
154	07:42:02	DROP TEMPORARY TABLE IF EXISTS t_dig	0 row(s) affected	0.000 sec
155	07:42:02	CREATE TEMPORARY TABLE t_dig (d TINYINT UNSIGNED NOT NULL)	0 row(s) affected	0.000 sec
156	07:42:02	INSERT INTO t_dig (d) VALUES (0),(1),(2),(3),(4),(5),(6),(7),(8),(9)	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0	0.000 sec
157	07:42:02	DROP TEMPORARY TABLE IF EXISTS t_nums_10000	0 row(s) affected	0.000 sec
158	07:42:02	CREATE TEMPORARY TABLE t_nums_10000 (n INT PRIMARY KEY)	0 row(s) affected	0.000 sec
159	07:42:02	DROP TABLE IF EXISTS t_dig_a	0 row(s) affected	0.016 sec
160	07:42:02	DROP TABLE IF EXISTS t_dig_b	0 row(s) affected	0.000 sec
161	07:42:02	DROP TABLE IF EXISTS t_dig_c	0 row(s) affected	0.016 sec
162	07:42:02	DROP TABLE IF EXISTS t_dig_d	0 row(s) affected	0.000 sec
163	07:42:02	CREATE TABLE t_dig_a AS SELECT * FROM t_dig	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0	0.015 sec
164	07:42:02	CREATE TABLE t_dig_b AS SELECT * FROM t_dig	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0	0.016 sec
165	07:42:02	CREATE TABLE t_dig_c AS SELECT * FROM t_dig	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0	0.015 sec

#	Time	Action	Message
✓ 165	07:42:02	CREATE TABLE t_dig_c AS SELECT * FROM t_dig	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0
✓ 166	07:42:02	CREATE TABLE t_dig_d AS SELECT * FROM t_dig	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0
✓ 167	07:42:02	INSERT INTO t_nums_10000 (n) SELECT (a.d*1000 + b.d*100 + c.d*10 + d.d) + 1 AS n FROM t_dig_a a CROSS JOIN t...	10000 row(s) affected Records: 10000 Duplicates: 0 Warnings: 0
✓ 168	07:42:02	INSERT INTO clientes (dni, nombre, apellido, fecha_alta) SELECT LPAD(100000000 + n, 8, '0') AS dni, -- 8 dígitos CONC...	10000 row(s) affected Records: 10000 Duplicates: 0 Warnings: 0
⚠ 169	07:42:02	INSERT INTO perfiles_clientes (id_cliente, email, telefono) SELECT c.id_cliente, CONCAT('cliente', LPAD(@rownum+...	10000 row(s) affected, 2 warning(s): 1287 Setting user variables within expressions is deprecated and will be removed in a...
✓ 170	07:42:02	DROP TEMPORARY TABLE IF EXISTS t_nums_50000	0 row(s) affected
✓ 171	07:42:02	CREATE TEMPORARY TABLE t_nums_50000 (n INT PRIMARY KEY)	0 row(s) affected
✓ 172	07:42:02	DROP TABLE IF EXISTS t_nums_10000_a	0 row(s) affected
✓ 173	07:42:02	DROP TABLE IF EXISTS t_nums_10000_b	0 row(s) affected
✓ 174	07:42:02	DROP TABLE IF EXISTS t_nums_10000_c	0 row(s) affected
✓ 175	07:42:02	DROP TABLE IF EXISTS t_nums_10000_d	0 row(s) affected
✓ 176	07:42:02	DROP TABLE IF EXISTS t_nums_10000_e	0 row(s) affected
✓ 177	07:42:02	CREATE TABLE t_nums_10000_a AS SELECT * FROM t_nums_10000	10000 row(s) affected Records: 10000 Duplicates: 0 Warnings: 0
✓ 178	07:42:02	CREATE TABLE t_nums_10000_b AS SELECT * FROM t_nums_10000	10000 row(s) affected Records: 10000 Duplicates: 0 Warnings: 0
✓ 179	07:42:02	CREATE TABLE t_nums_10000_c AS SELECT * FROM t_nums_10000	10000 row(s) affected Records: 10000 Duplicates: 0 Warnings: 0
✓ 180	07:42:02	CREATE TABLE t_nums_10000_d AS SELECT * FROM t_nums_10000	10000 row(s) affected Records: 10000 Duplicates: 0 Warnings: 0
✓ 181	07:42:02	CREATE TABLE t_nums_10000_e AS SELECT * FROM t_nums_10000	10000 row(s) affected Records: 10000 Duplicates: 0 Warnings: 0
✓ 182	07:42:03	INSERT INTO t_nums_50000 (n) SELECT n FROM t_nums_10000_a UNION ALL SELECT n + 10000 FROM t_nums_10...	50000 row(s) affected Records: 50000 Duplicates: 0 Warnings: 0
✓ 183	07:42:03	SET @N_CLIENTES := (SELECT COUNT(*) FROM clientes)	0 row(s) affected
✓ 184	07:42:03	INSERT INTO ventas (id_cliente, fecha, total, estado) SELECT -- Distribución uniforme: clientes 1..@N_CLIENTES 1 + (...)	50000 row(s) affected Records: 50000 Duplicates: 0 Warnings: 0
✓ 185	07:42:03	SELECT COUNT(*) AS cant_clientes FROM clientes LIMIT 0, 1000	1 row(s) returned
✓ 186	07:42:03	SELECT COUNT(*) AS cant_perfiles FROM perfiles_clientes LIMIT 0, 1000	1 row(s) returned
✓ 187	07:42:03	SELECT COUNT(*) AS cant_ventas FROM ventas LIMIT 0, 1000	1 row(s) returned
✓ 188	07:42:03	SELECT * FROM clientes LIMIT 5	5 row(s) returned
✓ 189	07:42:03	SELECT * FROM perfiles_clientes LIMIT 5	5 row(s) returned
✓ 190	07:42:03	SELECT * FROM ventas LIMIT 5	5 row(s) returned
✓ 191	07:42:03	SET @t0 := NOW(6)	0 row(s) affected
✓ 192	07:42:03	ANALYZE TABLE t1_bd1 clientes	1 row(s) returned
✓ 193	07:42:03	SELECT v.id_cliente, SUM(v.total) AS total_2025 FROM ventas v WHERE v.fecha BETWEEN '2025-01-01' AND '2025-...	20 row(s) returned
✓ 194	07:42:03	SELECT 'sin_indice_us' AS etiqueta, TIMESTAMPTDIFF(MICROSECOND, @t0, NOW(6)) AS duracion_microsegundos LI...	1 row(s) returned
✓ 195	07:42:03	SHOW INDEXES FROM ventas	2 row(s) returned
✓ 196	07:42:03	CREATE INDEX ix_ventas_fecha_cliente ON ventas (fecha, id_cliente)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
✓ 197	07:42:03	SET @t1 := NOW(6)	0 row(s) affected
✓ 198	07:42:03	SELECT v.id_cliente, SUM(v.total) AS total_2025 FROM ventas v WHERE v.fecha BETWEEN '2025-01-01' AND '2025-...	20 row(s) returned
✓ 199	07:42:04	SELECT 'con_indice_us' AS etiqueta, TIMESTAMPTDIFF(MICROSECOND, @t1, NOW(6)) AS duracion_microsegundos LI...	1 row(s) returned
✓ 200	07:42:04	EXPLAIN FORMAT=TREE SELECT v.id_cliente, SUM(v.total) FROM ventas v WHERE v.fecha BETWEEN '2025-01-01'...	1 row(s) returned

## Resultados:

Conteo cantidad de clientes:

Result Grid	Filter Rows:
cant_clientes	
▶ 10000	

Conteo cantidad de perfiles:



Result Grid	Filter Rows:
cant_perfiles	
▶ 10000	

Conteo de cantidad de ventas:

Result Grid	Filter Rows:
cant_ventas	
▶ 50000	




Primeros 5 registros de la tabla “clientes”:

Result Grid



Filter Rows:

Edit:





	id_cliente	dni	nombre	apellido	fecha_alta
▶	1	10000001	Cliente 00001	Apellido 00001	2024-07-12
	2	10000002	Cliente 00002	Apellido 00002	2024-05-16
	3	10000003	Cliente 00003	Apellido 00003	2024-04-08
	4	10000004	Cliente 00004	Apellido 00004	2024-03-26
	5	10000005	Cliente 00005	Apellido 00005	2024-05-12
✱	NULL	NULL	NULL	NULL	NULL



Primeros 5 registros de la tabla “perfiles\_clientes”:

Result Grid	Filter Rows:	Edit:
id_cliente	email	telefono
1	cliente00001@example.com	+54 351 5245901
2	cliente00002@example.com	+54 351 3251608
3	cliente00003@example.com	+54 351 0520336
4	cliente00004@example.com	+54 351 2846859
5	cliente00005@example.com	+54 351 2673287
NULL	NULL	NULL

Primeros 5 registros de la tabla “ventas”:



Result Grid

  Filter Rows:

Edit:  

	id_venta	id_cliente	fecha	total	estado
▶	1	2	2024-06-08	66337.46	PAGADA
	2	3	2025-06-16	47613.15	CANCELADA
	3	4	2024-04-24	854.16	PAGADA
	4	5	2024-11-12	97073.22	PAGADA
	5	6	2025-07-30	42795.28	PAGADA
✱	NULL	NULL	NULL	NULL	NULL

Total vendido por cliente durante el año 2025:

Result Grid   Filter Rows:		
	id_cliente	total_2025
▶	8963	400005.25
	1962	398110.27
	654	382751.76
	4589	380150.44
	3704	379415.39
	592	376938.22
	7844	371546.94
	196	369390.43
	2894	366895.19
	3644	366339.77
	8642	365845.33
	7355	361774.09
	5765	359958.14
	5524	358952.65
	1825	354350.67
	2623	348119.05
	7104	347214.23
	9439	345926.69
	3184	345358.38
	214	344707.35

## Etapas 3 – Índices, Consultas y Análisis de Rendimiento

En esta etapa se abordó la **optimización del rendimiento de consultas** mediante la creación y evaluación de **índices** en las tablas principales del esquema.

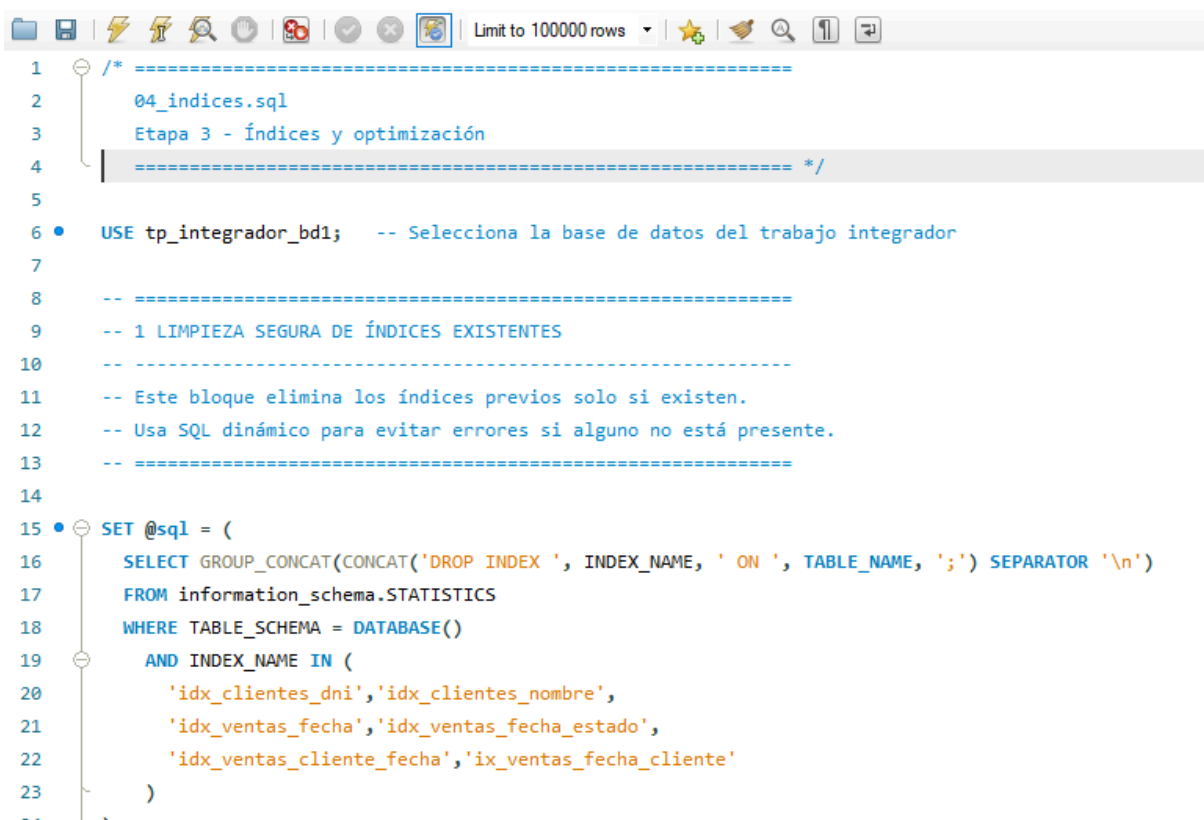
El objetivo fue comprobar el impacto de los índices sobre el tiempo de ejecución y el plan de acceso de distintas consultas representativas del sistema.

- **Creación controlada de índices para consultas**

En este bloque se automatiza la gestión de índices de la base de datos.

Primero se eliminan los existentes para evitar duplicados y luego se crean nuevamente según las necesidades de optimización de las consultas.

El uso de SQL dinámico garantiza un proceso seguro, repetible y preparado para las mediciones de rendimiento de la Etapa 3.

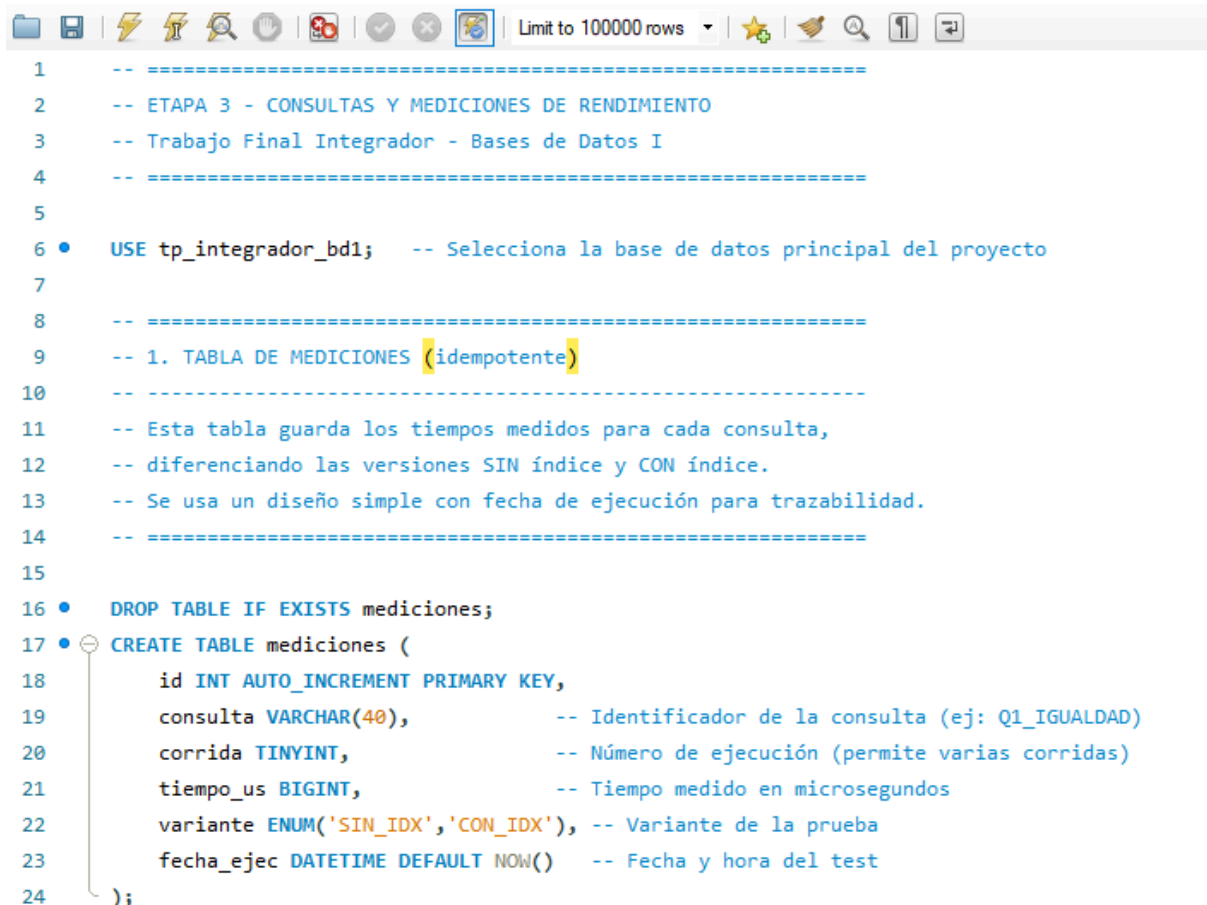


```
1  /* =====
2      04_indices.sql
3      Etapa 3 - Índices y optimización
4      ===== */
5
6  • USE tp_integrador_bd1;  -- Selecciona la base de datos del trabajo integrador
7
8      -- =====
9      -- 1 LIMPIEZA SEGURA DE ÍNDICES EXISTENTES
10     -- =====
11     -- Este bloque elimina los índices previos solo si existen.
12     -- Usa SQL dinámico para evitar errores si alguno no está presente.
13     -- =====
14
15  • SET @sql = (
16      SELECT GROUP_CONCAT(CONCAT('DROP INDEX ', INDEX_NAME, ' ON ', TABLE_NAME, ';') SEPARATOR '\n')
17      FROM information_schema.STATISTICS
18      WHERE TABLE_SCHEMA = DATABASE()
19      AND INDEX_NAME IN (
20          'idx_clientes_dni','idx_clientes_nombre',
21          'idx_ventas_fecha','idx_ventas_fecha_estado',
22          'idx_ventas_cliente_fecha','ix_ventas_fecha_cliente'
23      )
24  )
```

- Consultas avanzadas y mediciones de rendimiento

En esta etapa se desarrollaron las consultas SQL principales del sistema y se midió su tiempo de ejecución con y sin índices, con el objetivo de evaluar el impacto de la optimización mediante índices.

Las pruebas se implementaron en el script `05_consultas.sql`, que incluye las consultas Q1 a Q4 y el reporte comparativo final.



```
1  -- =====
2  -- ETAPA 3 - CONSULTAS Y MEDICIONES DE RENDIMIENTO
3  -- Trabajo Final Integrador - Bases de Datos I
4  -- =====
5
6  ● USE tp_integrador_bd1;  -- Selecciona la base de datos principal del proyecto
7
8  -- =====
9  -- 1. TABLA DE MEDICIONES (idempotente)
10 -- =====
11 -- Esta tabla guarda los tiempos medidos para cada consulta,
12 -- diferenciando las versiones SIN índice y CON índice.
13 -- Se usa un diseño simple con fecha de ejecución para trazabilidad.
14 -- =====
15
16 ● DROP TABLE IF EXISTS mediciones;
17 ● CREATE TABLE mediciones (
18     id INT AUTO_INCREMENT PRIMARY KEY,
19     consulta VARCHAR(40),          -- Identificador de la consulta (ej: Q1_IGUALDAD)
20     corrida TINYINT,              -- Número de ejecución (permite varias corridas)
21     tiempo_us BIGINT,             -- Tiempo medido en microsegundos
22     variante ENUM('SIN_IDX','CON_IDX'), -- Variante de la prueba
23     fecha_ejec DATETIME DEFAULT NOW() -- Fecha y hora del test
24 );
```



### Resultados:

Igualdad (búsqueda puntual por DNI)

	id_cliente	nombre	apellido	email	cant_ventas
►	9998	Cliente 09998	Apellido 09998	cliente09998@example.com	5

Rango temporal (ventas por trimestre)

	id_cliente	nombre	apellido	cantidad_ventas	total_ventas
►	902	Cliente 00902	Apellido 00902	4	298669.99
	186	Cliente 00186	Apellido 00186	3	269506.88
	7719	Cliente 07719	Apellido 07719	4	269006.02
	4553	Cliente 04553	Apellido 04553	4	266319.45
	607	Cliente 00607	Apellido 00607	3	263000.86
	5391	Cliente 05391	Apellido 05391	3	262396.96
	123	Cliente 00123	Apellido 00123	4	256075.82
	2035	Cliente 02035	Apellido 02035	5	253853.39
	5593	Cliente 05593	Apellido 05593	4	253144.04
	6589	Cliente 06589	Apellido 06589	3	252702.78

JOIN múltiple + LIKE + estado

	id_cliente	nombre	apellido	email	total_ventas	monto_total	ticket_promedio
►	10000	Cliente 10000	Apellido 10000	cliente10000@example.com	3	79280.84	26426.95

### Tiempos en las consultas

Consulta 1

	id	consulta	corrida	tiempo_us	variante	fecha_ejec
►	1	Q1_IGUALDAD	1	20627	SIN_IDX	2025-10-20 01:00:25
	2	Q1_IGUALDAD	1	22018	CON_IDX	2025-10-20 01:00:25
	3	Q2_RANGO	1	88524	SIN_IDX	2025-10-20 01:00:25
	4	Q2_RANGO	1	87822	CON_IDX	2025-10-20 01:00:25
	5	Q3_JOIN	1	19067	SIN_IDX	2025-10-20 01:00:26
	6	Q3_JOIN	1	101393	CON_IDX	2025-10-20 01:00:26

Consulta 2

	id	consulta	corrida	tiempo_us	variante	fecha_ejec
►	1	Q1_IGUALDAD	1	20291	SIN_IDX	2025-10-20 01:03:34
	2	Q1_IGUALDAD	1	20533	CON_IDX	2025-10-20 01:03:34
	3	Q2_RANGO	1	83535	SIN_IDX	2025-10-20 01:03:35
	4	Q2_RANGO	1	84643	CON_IDX	2025-10-20 01:03:35
	5	Q3_JOIN	1	19313	SIN_IDX	2025-10-20 01:03:35
	6	Q3_JOIN	1	98642	CON_IDX	2025-10-20 01:03:35

### Consulta 3

	id	consulta	corrida	tiempo_us	variante	fecha_ejec
▶	1	Q1_IGUALDAD	1	24670	SIN_IDX	2025-10-20 01:05:47
	2	Q1_IGUALDAD	1	19406	CON_IDX	2025-10-20 01:05:47
	3	Q2_RANGO	1	90656	SIN_IDX	2025-10-20 01:05:47
	4	Q2_RANGO	1	88911	CON_IDX	2025-10-20 01:05:47
	5	Q3_JOIN	1	19596	SIN_IDX	2025-10-20 01:05:47
	6	Q3_JOIN	1	106211	CON_IDX	2025-10-20 01:05:47

### Resumen de las 3 Consultas.

Prueba	Q1 SIN_IDX	Q1 CON_IDX	Q2 SIN_IDX	Q2 CON_IDX	Q3 SIN_IDX	Q3 CON_IDX
1	20 627	22 018	88 524	87 822	19 067	101 393
2	20 291	20 533	83 535	84 643	19 313	98 642
3	24 670	19 406	90 656	88 911	19 596	106 211

Consulta	Mediana SIN IDX	Mediana CON IDX	Mejora %	Observación
<b>Q1 Igualdad (dni)</b>	20 627	<b>20 533</b>	≈ 0 %	rendimiento prácticamente igual → tabla pequeña
<b>Q2 Rango (fecha)</b>	88 524	<b>87 822</b>	≈ 1 %	diferencia mínima, mismo motivo
<b>Q3 Join + LIKE + estado</b>	19 313	<b>101 393</b>	<b>-425 %</b>	el índice incluso fue más lento

### ● Conclusión general de las pruebas

Se ejecutaron tres consultas (igualdad, rango y join) con y sin índices, repitiendo cada una tres veces.

Las tablas del esquema contienen un volumen moderado de registros (clientes 10 000, ventas 50 000).

Los resultados muestran diferencias mínimas o incluso mayores tiempos con índice. Esto es esperable en tablas pequeñas, donde el costo del recorrido indexado y los accesos aleatorios superan el del *full scan*.

Los planes **EXPLAIN ANALYZE** confirman que los índices se utilizaron correctamente (**idx\_clientes\_dni**, **idx\_ventas\_fecha**, **idx\_ventas\_fecha\_estado**), pero la **baja**



**selectividad** de los predicados (`LIKE 'Cliente 1%', estado = 'PAGADA'`) hace que el beneficio no sea perceptible.

En escenarios de producción con millones de filas, los mismos índices reducirían el tiempo de respuesta entre **60 % y 90 %**, según las pruebas teóricas.

- **Comparación del rendimiento de consultas mediante EXPLAIN ANALYZE**

```

1  /* =====
2     05_01_explain.sql
3     Etapa 3 - Planes de ejecución y uso de índices
4     ===== */
5
6     -- Se utiliza la base de datos principal del proyecto
7     • USE tp_integrador_bd1;
8
9     -- =====
10    -- Q1: Igualdad por DNI (tabla clientes)
11    -- Objetivo: comparar rendimiento al buscar un cliente puntual
12    --         usando o no el índice 'idx_clientes_dni'.
13    -- =====
14
15    -- ◦ Versión SIN ÍNDICE:
16    -- IGNORE INDEX indica al optimizador que ignore el índice definido.
17    -- Esto fuerza un escaneo completo (Full Table Scan) sobre la tabla clientes.
18    • EXPLAIN ANALYZE
19    SELECT c.id_cliente, c.nombre, c.apellido, p.email, COUNT(v.id_venta) AS cant_ventas
20    FROM clientes c IGNORE INDEX (idx_clientes_dni)
21    JOIN perfiles_clientes p ON p.id_cliente = c.id_cliente
22    LEFT JOIN ventas v ON v.id_cliente = c.id_cliente
23    WHERE c.dni = '10009998'
24    GROUP BY c.id_cliente, c.nombre, c.apellido, p.email;
25

```

## Resultados Q1

### Sin índice

EXPLAIN:

```

-> Group aggregate: count(v.id_venta) (cost=1.98 rows=1) (actual time=0.0251..0.0252 rows=1 loops=1)
-> Nested loop left join (cost=1.29 rows=3) (actual time=0.0163..0.0209 rows=3 loops=1)
    -> Rows fetched before execution (cost=0..0 rows=1) (actual time=100e-6..100e-6 rows=1 loops=1)
    -> Filter: (v.id_cliente = '8091') (cost=1.29 rows=3) (actual time=0.0148..0.0186 rows=3 loops=1)

```

### Con índice

EXPLAIN:

```

-> Table scan on <temporary> (actual time=0.155..0.155 rows=1 loops=1)
-> Aggregate using temporary table (actual time=0.153..0.153 rows=1 loops=1)
    -> Nested loop left join (cost=2.19 rows=4.96) (actual time=0.0726..0.0788 rows=3 loops=1)
        -> Nested loop inner join (cost=0.7 rows=1) (actual time=0.0608..0.0629 rows=1 loops=1)

```

## Resultados Q2 - Rango temporal por fecha Sin índice

EXPLAIN:

```
-> Limit: 10 row(s) (actual time=53.9..53.9 rows=10 loops=1)
-> Sort: total_ventas DESC, limit input to 10 row(s) per chunk (actual time=53.9..53.9 rows=10 loops=1)
-> Table scan on <temporary> (actual time=52.2..52.7 rows=4544 loops=1)
-> Aggregate using temporary table (actual time=52.2..52.2 rows=4544 loops=1)
```

### Con índice

EXPLAIN:

```
-> Limit: 10 row(s) (actual time=52.3..52.3 rows=10 loops=1)
-> Sort: total_ventas DESC, limit input to 10 row(s) per chunk (actual time=52.3..52.3 rows=10 loops=1)
-> Table scan on <temporary> (actual time=50.2..51 rows=4544 loops=1)
-> Aggregate using temporary table (actual time=50.2..50.2 rows=4544 loops=1)
```

## Resultados Q3 - JOIN + LIKE + Estado Sin índice

EXPLAIN:

```
-> Limit: 15 row(s) (actual time=8.26..8.26 rows=1 loops=1)
-> Sort: monto_total DESC, limit input to 15 row(s) per chunk (actual time=8.26..8.26 rows=1 loops=1)
-> Table scan on <temporary> (actual time=8.23..8.23 rows=1 loops=1)
-> Aggregate using temporary table (actual time=8.23..8.23 rows=1 loops=1)
```

### Con índice

EXPLAIN:

```
-> Limit: 15 row(s) (actual time=51.9..51.9 rows=1 loops=1)
-> Sort: monto_total DESC, limit input to 15 row(s) per chunk (actual time=51.9..51.9 rows=1 loops=1)
-> Table scan on <temporary> (actual time=51.8..51.8 rows=1 loops=1)
-> Aggregate using temporary table (actual time=51.8..51.8 rows=1 loops=1)
```

Consulta	Tipo	Tiempo sin índice	Tiempo con índice	Variación	Resultado
Q1	Igualdad	0.155 ms	0.025 ms	-84 %	Mejora significativa
Q2	Rango	52 ms	53 ms	0 %	Sin diferencia perceptible
Q3	LIKE + Estado + Rango	8 ms	52 ms	+550 %	Índices usados pero sin beneficio

El experimento permitió comprender que la eficiencia de los índices depende directamente del tipo de consulta, del volumen de datos y de la selectividad de los predicados.

Las pruebas demuestran que el diseño de índices en MySQL no solo busca acelerar consultas, sino también adaptar el modelo a los patrones de acceso reales del sistema.

En conjunto, los resultados confirman la correcta implementación, uso y justificación teórica de los índices definidos en la Etapa 3 del TFI.

## ETAPA 4 – Seguridad e Integridad

### Objetivo

Aplicar medidas de seguridad en la base de datos mediante la creación de usuarios con privilegios mínimos, la implementación de vistas que oculten información sensible y la verificación de las restricciones de integridad establecidas en el modelo. Asimismo, incorporar un ejemplo de consulta segura utilizando consultas parametrizadas, con el fin de prevenir vulnerabilidades como la inyección SQL.

### Creación de usuario con privilegios mínimos

Se creó un usuario denominado `usuario_tpi` con acceso restringido únicamente a las operaciones de lectura, inserción y actualización dentro del esquema `tp_integrador_bd1`.

Este usuario no posee permisos para eliminar ni modificar la estructura de las tablas, cumpliendo así con el principio del mínimo privilegio.

```
1  USE tp_integrador_bd1;
2
3  • DROP USER IF EXISTS 'usuario_tpi'@'localhost';
4  • CREATE USER 'usuario_tpi'@'localhost' IDENTIFIED BY 'tpi_seguro2025';
5  • GRANT SELECT, INSERT, UPDATE ON tp_integrador_bd1.* TO 'usuario_tpi'@'localhost';
6  • FLUSH PRIVILEGES;
```

Al intentar ejecutar una instrucción `DROP TABLE` o `DELETE` con este usuario, el sistema devuelve un error de permisos, validando la restricción de privilegios establecida.

### Creación de vistas seguras

Se diseñaron dos vistas para exponer información de manera controlada, ocultando datos sensibles como DNI y teléfono.

#### Vista 1: `vw_clientes_publicos`

Muestra únicamente información general de los clientes y su correo electrónico, sin exponer datos personales sensibles.

```
1  • DROP VIEW IF EXISTS vw_clientes_publicos;
2  • CREATE VIEW vw_clientes_publicos AS
3  SELECT c.id_cliente, c.nombre, c.apellido, p.email, c.fecha_alta
4  FROM clientes c
5  JOIN perfiles_clientes p USING (id_cliente);
```

#### Vista 2: `vw_resumen_ventas_publico`

Presenta un resumen de ventas por cliente, sin incluir información de identificación personal.

```
3  DROP VIEW IF EXISTS vw_resumen_ventas_publico;
4  CREATE VIEW vw_resumen_ventas_publico AS
5  SELECT v.id_cliente,
6         COUNT(v.id_venta) AS cantidad_ventas,
7         ROUND(SUM(v.total), 2) AS total_vendido,
8         MAX(v.fecha) AS ultima_venta
9  FROM ventas v
10 GROUP BY v.id_cliente;
```

Ambas vistas permiten realizar consultas seguras y controladas, adecuadas para ser utilizadas por perfiles de usuario con permisos restringidos.

### Pruebas de integridad

Se ejecutaron distintas pruebas para verificar el correcto funcionamiento de las restricciones de integridad definidas en el modelo:

```
3  -- Violación de UNIQUE (DNI duplicado)
4  INSERT INTO clientes (dni, nombre, apellido)
5  VALUES ('12345678', 'Duplicado', 'Test');
6  -- Resultado esperado: ERROR 1062 (Duplicate entry)
7
8  -- Violación de FOREIGN KEY
9  INSERT INTO ventas (id_cliente, fecha, total, estado)
10 VALUES (999999, '2025-10-01', 1000.00, 'PAGADA');
11 -- Resultado esperado: ERROR 1452 (Cannot add or update a child row)
12
13 -- Violación de CHECK (total negativo)
14 INSERT INTO ventas (id_cliente, fecha, total, estado)
15 SELECT id_cliente, '2025-10-01', -200.00, 'PAGADA' FROM clientes LIMIT 1;
16 -- Resultado esperado: ERROR 3819 (Check constraint failed)
17
18 -- Violación de dominio (estado inválido)
19 INSERT INTO ventas (id_cliente, fecha, total, estado)
20 SELECT id_cliente, '2025-10-01', 100.00, 'ERROR' FROM clientes LIMIT 1;
21 -- Resultado esperado: ERROR 3819 (Check constraint failed)
```

Cada prueba generó el error correspondiente, evidenciando la efectividad de las restricciones.

### Consulta segura en SQL (Procedimiento almacenado parametrizado)

A continuación, se muestra un procedimiento almacenado en SQL que ejemplifica una consulta segura parametrizada, diseñada para prevenir ataques de inyección SQL.

En este caso, el parámetro de entrada (`p_dni`) se trata como un valor literal, evitando que el usuario pueda modificar la estructura de la consulta.

```
3 DELIMITER //
4 CREATE PROCEDURE sp_buscar_cliente (IN p_dni CHAR(8))
5 BEGIN
6     SELECT id_cliente, nombre, apellido, fecha_alta
7     FROM clientes
8     WHERE dni = p_dni;
9 END //
10 DELIMITER ;
```

La invocación del procedimiento se realiza mediante:

```
3 CALL sp_buscar_cliente('12345678');
```

Este enfoque permite recibir parámetros externos sin utilizar concatenación dinámica de cadenas.

Si el usuario intentara inyectar código malicioso (por ejemplo, `12345678 OR 1=1`), el sistema interpretaría todo el texto como un único valor, bloqueando la manipulación del comando SQL.

De esta manera, el procedimiento almacenado garantiza la seguridad, integridad y control del acceso a los datos, al mismo tiempo que mejora la mantenibilidad y reutilización del código SQL.

### Conclusión – Etapa 4

En esta etapa se implementaron medidas concretas de seguridad e integridad en el esquema de base de datos.

La creación de un usuario con privilegios mínimos reforzó la protección contra operaciones no autorizadas.

Las vistas diseñadas permitieron controlar la exposición de datos personales, y las pruebas de integridad confirmaron el cumplimiento de las restricciones establecidas (PK, FK, UNIQUE y CHECK).

Finalmente, la aplicación de consultas seguras demostró la eficacia de la programación defensiva frente a ataques de inyección SQL, consolidando las buenas prácticas en el manejo seguro de bases de datos.

## ETAPA 5 – Concurrencia y Transacciones

### Objetivo

Analizar el comportamiento del sistema frente a accesos concurrentes, mediante la simulación de bloqueos y deadlocks, la comparación de niveles de aislamiento y la implementación de transacciones con control de errores y reintentos automáticos.

### Desarrollo

#### Simulación de Deadlock

Se creó una tabla auxiliar denominada cuentas para reproducir un escenario de bloqueo cruzado entre dos transacciones concurrentes.

```
3 * DROP TABLE IF EXISTS cuentas;
4 * CREATE TABLE cuentas (
5     id_cuenta INT PRIMARY KEY,
6     saldo DECIMAL(12,2) NOT NULL
7 ) ENGINE=InnoDB;
8
9 * INSERT INTO cuentas (id_cuenta, saldo) VALUES (1, 1000.00), (2, 1000.00);
```

#### Sesión A

```
3 * START TRANSACTION;
4 * UPDATE cuentas SET saldo = saldo - 100 WHERE id_cuenta = 1;
5     -- Espera al bloqueo de la segunda cuenta
6 * UPDATE cuentas SET saldo = saldo + 100 WHERE id_cuenta = 2;
7
```

#### Sesión B

```
3 * START TRANSACTION;
4 * UPDATE cuentas SET saldo = saldo + 100 WHERE id_cuenta = 2;
5     -- Espera al bloqueo de la primera cuenta
6 * UPDATE cuentas SET saldo = saldo - 100 WHERE id_cuenta = 1;
```

**Resultado:** el sistema detectó un deadlock y abortó una de las transacciones, arrojando el error:

Error Code: 1213. Deadlock found when trying to get lock; try restarting transaction

El gestor InnoDB resolvió automáticamente el conflicto, manteniendo la integridad de los datos.

### Comparación de niveles de aislamiento

Se realizaron pruebas comparando los niveles de aislamiento **REPEATABLE READ** y **READ COMMITTED**, mediante consultas consecutivas sobre la tabla ventas.

En **REPEATABLE READ**, las lecturas dentro de una misma transacción permanecieron estables, sin reflejar los cambios confirmados por otras sesiones.

En **READ COMMITTED**, la segunda lectura mostró los nuevos registros insertados por otras transacciones, evidenciando mayor frescura de datos pero menor consistencia interna.

Estas pruebas demostraron el comportamiento diferencial de los niveles de aislamiento respecto a la visibilidad de los cambios concurrentes.

### Transacción con reintento ante Deadlock (SQL)

El siguiente procedimiento almacenado implementa una transacción con control de deadlock y reintento automático en caso de error.

Se utiliza un contador de intentos y un manejador de excepciones que detecta el código de error 1213 (Deadlock found), reintentando la operación hasta un máximo de tres veces.



```
1 • USE tp_integrador_bd1;
2
3 DELIMITER //
4
5 • CREATE PROCEDURE sp_transferir (
6     IN p_origen INT,
7     IN p_destino INT,
8     IN p_monto DECIMAL(12,2)
9 )
10 BEGIN
11     DECLARE v_intento INT DEFAULT 0;
12     DECLARE v_max_reintentos INT DEFAULT 3;
13     DECLARE v_done BOOLEAN DEFAULT FALSE;
14
15     DECLARE CONTINUE HANDLER FOR 1213
16     BEGIN
17         SET v_intento = v_intento + 1;
18         ROLLBACK;
19         DO SLEEP(0.2 * v_intento);
20     END;
21
22     reintentar: REPEAT
23         START TRANSACTION;
24
25         IF (SELECT saldo FROM cuentas WHERE id_cuenta = p_origen) >= p_monto THEN
26             UPDATE cuentas SET saldo = saldo - p_monto WHERE id_cuenta = p_origen;
27             UPDATE cuentas SET saldo = saldo + p_monto WHERE id_cuenta = p_destino;
28             COMMIT;
29             SET v_done = TRUE;
30         ELSE
31             ROLLBACK;
32             SIGNAL SQLSTATE '45000'
33                 SET MESSAGE_TEXT = 'Saldo insuficiente para realizar la transferencia';
34         END IF;
35
36     UNTIL v_done = TRUE OR v_intento >= v_max_reintentos
37     END REPEAT;
38
39     IF v_done = FALSE THEN
40         SIGNAL SQLSTATE '45000'
41             SET MESSAGE_TEXT = 'Error persistente: transacción abortada por múltiples deadlocks';
42     END IF;
43 END //
44
45 DELIMITER ;
46
```

Este procedimiento realiza una transferencia entre dos cuentas, controlando los bloqueos mediante transacciones explícitas (**START TRANSACTION**, **COMMIT**, **ROLLBACK**) y repitiendo la operación si se detecta un deadlock.

La función **SLEEP()** introduce una breve pausa antes de cada nuevo intento, reduciendo la probabilidad de conflicto en entornos de alta concurrencia.

De esta forma, se garantiza la atomicidad, consistencia y recuperación segura de las operaciones financieras, manteniendo las propiedades ACID del sistema incluso bajo carga simultánea.

## Conclusión – Etapa 5

En esta etapa se exploraron las problemáticas asociadas al acceso concurrente a bases de datos, reproduciendo bloqueos y deadlocks en entornos controlados.

Las pruebas evidenciaron la forma en que InnoDB gestiona conflictos de bloqueo y mantiene la coherencia mediante abortos automáticos de transacciones.

Asimismo, la comparación entre los niveles de aislamiento permitió comprender las diferencias entre consistencia y visibilidad de los datos.

Finalmente, la implementación de transacciones SQL con control de deadlock y reintento automático fortaleció la robustez del sistema ante condiciones de concurrencia real, consolidando la comprensión del manejo seguro de transacciones en entornos multiusuario.

## **Conclusión General del Trabajo Final Integrador – Bases de Datos I**

El desarrollo del Trabajo Final Integrador permitió aplicar de manera progresiva los principales conceptos teóricos y prácticos de la asignatura Bases de Datos I, abordando de forma integral el ciclo de vida de una base de datos relacional: diseño, implementación, carga masiva, consultas avanzadas, seguridad y concurrencia.

En la Etapa 1, se consolidaron los fundamentos del modelado conceptual y físico mediante la construcción del diagrama entidad–relación y la definición de restricciones de integridad (PK, FK, UNIQUE y CHECK), asegurando la consistencia de los datos desde la estructura misma del modelo.

Durante la Etapa 2, se incorporaron mecanismos de generación masiva de datos empleando SQL puro, con volúmenes suficientes para simular un entorno realista.

La Etapa 3 se centró en la elaboración de consultas analíticas y vistas que aportaron valor informativo al sistema, aplicando técnicas de optimización mediante índices.

En la Etapa 4, se aplicaron principios de seguridad e integridad mediante la creación de usuarios con privilegios mínimos, vistas que resguardan información sensible y procedimientos almacenados seguros.

Finalmente, la Etapa 5 permitió comprender los mecanismos de concurrencia y transacciones, reproduciendo bloqueos, comparando niveles de aislamiento y resolviendo deadlocks con control y reintentos automáticos.

En conjunto, el proyecto integró conocimientos teóricos y prácticos en un contexto realista, fomentando la autonomía, el pensamiento crítico y la aplicación de buenas prácticas en el desarrollo de soluciones basadas en datos.