

Universidad Tecnológica Nacional

# Bases de datos I



**Grupo:**

- Lautaro Zullo
- Walter Verdún
- Ignacio Salazar

**Tutor:** Constanza Uño

**Profesor:** Cinthia Rigoni

# Bases de datos I

## ÍNDICE

<b>ÍNDICE.....</b>	<b>1</b>
<b>Resolución de consignas.....</b>	<b>3</b>
<b>Etapa 1 – Modelado y Definición de Constraints.....</b>	<b>3</b>
Diagrama EER.....	3
Inserción de 2 filas erróneas.....	3
Inserción de 2 filas validadas.....	4
Interacción con IA.....	4
Índices.....	4
<b>Etapa 2. Generación y carga de datos masivos con SQL puro.....</b>	<b>5</b>
Poblar la tabla credenciales.....	5
Poblar la tabla usuarios.....	6
<b>Etapa 3. Consultas complejas y útiles a partir del CRUD inicial.....</b>	<b>7</b>
Consulta para obtener nombre de usuario, email y hash de la contraseña.....	7
Consulta para obtener todos los usuarios editados posteriores a una fecha.....	7
Consulta para ver el porcentaje de usuarios que requieren reset por rol.....	7
Contar usuarios por rol, y mostrar solo los roles con más de 3000 usuarios.....	7
Tiempo de ejecución de las consultas.....	8
Comando EXPLAIN ANALYZE.....	8
Tiempos registrados.....	8
Tiempos registrados.....	9
Tiempos registrados.....	10
Tiempos registrados.....	11
Consultas de tiempo de ejecución con índice.....	12
Tiempos registrados.....	12
Comando EXPLAIN ANALYZE.....	12
Tiempos registrados.....	13
Comando EXPLAIN ANALYZE.....	13
Tiempos registrados.....	14
Comando EXPLAIN ANALYZE.....	14
Tiempos registrados.....	15
Comando EXPLAIN ANALYZE.....	15
Tiempos registrados.....	16
Comando EXPLAIN ANALYZE.....	16
<b>Etapa 4. Seguridad e integridad.....</b>	<b>17</b>
Creación de usuarios.....	17
Vista reporte seguridad.....	17
Creación de procedimiento.....	18
Caso correcto.....	19
Caso incorrecto.....	20
<b>Etapa 5. Conurrencia y transacciones.....</b>	<b>21</b>

# Bases de datos I

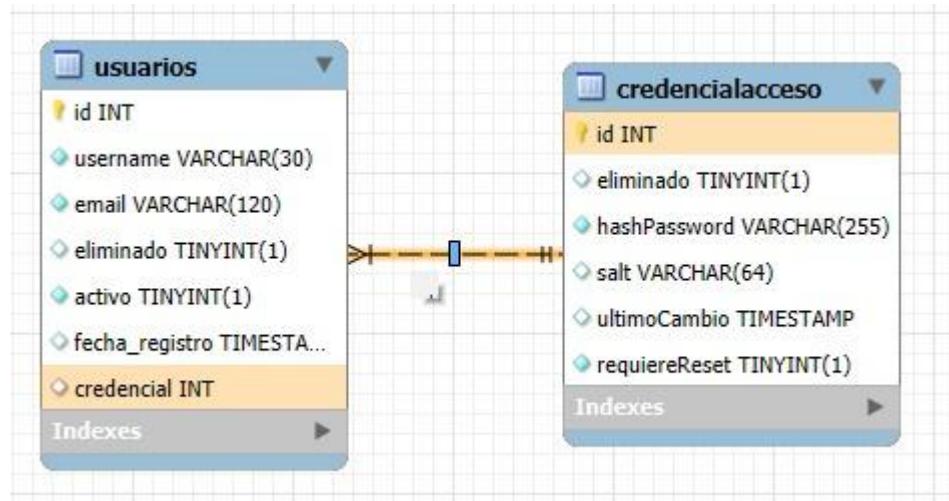
Deadlock.....	21
Deadlock con rollback.....	22
READ COMMITTED.....	23
REPEATABLE READ.....	23
<b>Conclusión.....</b>	<b>24</b>
<b>Video.....</b>	<b>24</b>

# Bases de datos I

## Resolución de consignas

### Etapa 1 - Modelado y Definición de Constraints

#### Diagrama EER



#### Inserción de 2 filas erróneas

```
1 • SELECT * FROM credencialesusuarios.usuarios;
2
3 • INSERT INTO Usuarios(username, email, eliminado, activo, rol, credencial_id) VALUES("Lauti", "zullolau@gmail.com", False, True, "SOY ADMIN", 1);
4 • INSERT INTO Usuarios(username, email, eliminado, activo, rol, credencial_id) VALUES("Walter", "walter@gmail.com", False, True, "MOD", 1);
5
6 • INSERT INTO Usuarios(username, email, eliminado, activo, rol) VALUES("Lauti", "zullolau@gmail.com", False, True, "Administrador");
7 • INSERT INTO Usuarios(username, email, eliminado, activo, rol, credencial_id) VALUES("Walter", "walter@gmail.com", False, True, "Moderador", 2);
```

Action	Time	Message
1 19:31:03 INSERT INTO Usuarios(username, email, eliminado, activo, rol, credencial_id) VALUES("Lauti", "zullolau@gmail.com", False, True, "SOY ADMIN", 1)	19:31:03	Error Code: 3819. Check constraint 'usuarios_chk_1' is violated.
2 19:31:03 INSERT INTO Usuarios(username, email, eliminado, activo, rol, credencial_id) VALUES("Walter", "walter@gmail.com", False, True, "MOD", 1)	19:31:03	Error Code: 3819. Check constraint 'usuarios_chk_1' is violated.
3 19:31:12 INSERT INTO Usuarios(username, email, eliminado, activo, rol) VALUES("Lauti", "zullolau@gmail.com", False, True, "Administrador")	19:31:12	Error Code: 1364. Field 'credencial_id' doesn't have a default value
4 19:31:15 INSERT INTO Usuarios(username, email, eliminado, activo, rol, credencial_id) VALUES("Walter", "walter@gmail.com", False, True, "Moderador", 2)	19:31:15	Error Code: 1062. Duplicate entry 'Walter' for key 'usuarios.username'

# Bases de datos I

## Inserción de 2 filas validadas

The screenshot shows the MySQL Workbench interface. At the top, a script editor window displays several SQL statements. Below it, an 'Output' window shows the results of the execution.

**Script (Top Window):**

```
1 •  SELECT * FROM credencialesusuarios.usuarios;
2
3 •  INSERT INTO Usuarios(username, email, eliminado, activo, rol, credencial_id) VALUES("Lauti", "zulollau@gmail.com", False, True, "SOY ADMIN", 1);
4 •  INSERT INTO Usuarios(username, email, eliminado, activo, rol, credencial_id) VALUES("Walter", "walter@gmail.com", False, True, "MOD", 1);
5
6 •  INSERT INTO Usuarios(username, email, eliminado, activo, rol, credencial_id) VALUES("Lauti", "zulollau@gmail.com", False, True, "Administrador", 1);
7 •  INSERT INTO Usuarios(username, email, eliminado, activo, rol, credencial_id) VALUES("Walter", "walter@gmail.com", False, True, "Moderador", 2);
```

**Action Output (Bottom Window):**

#	Time	Action	Message
3	19:27:11	INSERT INTO Usuarios(username, email, eliminado, activo, rol) VALUES("Lauti", "zulollau@gmail.com", False, True, "Administrador")	Error Code: 1364. Field 'credencial_id' doesn't have a default value
4	19:27:50	INSERT INTO Usuarios(username, email, eliminado, activo, rol, credencial_id) VALUES("Lauti", "zulollau@gmail.com", False, True, "Administrador", 1)	1 row(s) affected
5	19:28:07	INSERT INTO Credencialacceso(eliminado, hashPassword, salt, requiereReset) VALUES(False, "Jkkmdujnfcasjkdcnlaajkcrdas3", 12, False)	1 row(s) affected
6	19:28:08	SELECT * FROM credencialesusuarios.credencialacceso LIMIT 0, 1000	2 row(s) returned
7	19:28:22	INSERT INTO Usuarios(username, email, eliminado, activo, rol, credencial_id) VALUES("Walter", "walter@gmail.com", False, True, "Moderador", 2)	1 row(s) affected

## Interacción con IA

<https://chatgpt.com/share/68f6b7a9-4f28-8007-999b-01c6495f9418>

## Índices

```
CREATE INDEX idx_activo_eliminado ON Usuarios (activo, eliminado);
CREATE INDEX idx_rol ON Usuarios (rol);
CREATE INDEX idx_fecha_registro ON Usuarios (fecha_registro);

CREATE INDEX idx_requiereReset ON CredencialAcceso (requiereReset);
CREATE INDEX idx_ultimoCambio ON CredencialAcceso (ultimoCambio);
CREATE INDEX idx_eliminado ON CredencialAcceso (eliminado);
```

# Bases de datos I

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

### Poblar la tabla credenciales

```
DELIMITER //
CREATE PROCEDURE poblar_credenciales(IN cantidad INT)
BEGIN
    DECLARE i INT DEFAULT 1;
    WHILE i <= cantidad DO
        INSERT INTO CredencialAcceso (hashPassword, salt, requiereReset)
        VALUES (
            CONCAT('hash_', i),
            CONCAT('salt_', i),
            FALSE
        );
        SET i = i + 1;
    END WHILE;
END //
DELIMITER ;

CALL poblar_credenciales(200000);
```

# Bases de datos I

## Poblar la tabla usuarios

```
DELIMITER //
CREATE PROCEDURE poblar_usuarios(IN cantidad INT)
BEGIN
    DECLARE i INT DEFAULT 1;
    DECLARE offset_id INT DEFAULT 0;

    SELECT IFNULL(MAX(id), 0) INTO offset_id FROM Usuarios;

    WHILE i <= cantidad DO
        INSERT INTO Usuarios (username, email, rol, credencial_id)
        VALUES (
            CONCAT('usuario_', offset_id + i),
            CONCAT('usuario', offset_id + i, '@correo.com'),
            CASE
                WHEN (offset_id + i) % 100 = 0 THEN 'Administrador'
                WHEN (offset_id + i) % 10 = 0 THEN 'Moderador'
                ELSE 'Usuario'
            END,
            offset_id + i
        );
        SET i = i + 1;
    END WHILE; END //
DELIMITER ;
CALL poblar_usuarios(200000);
```

# Bases de datos I

## Etapa 3. Consultas complejas y útiles a partir del CRUD inicial

Consulta para obtener nombre de usuario, email y hash de la contraseña

```
SELECT u.username, u.email, ca.hashPassword FROM Usuarios AS u INNER JOIN CredencialAcceso AS ca ON u.credencial_id = ca.id;
```

Consulta para obtener todos los usuarios editados posteriores a una fecha

```
SELECT u.id, u.username, u.rol, ca.ultimoCambio FROM Usuarios AS u INNER JOIN CredencialAcceso AS ca ON u.credencial_id = ca.id WHERE ultimoCambio > "2025-10-20 19:43:33";
```

Consulta para ver el porcentaje de usuarios que requieren reset por rol

```
SELECT
    u.rol,
    COUNT(*) AS TotalPorRol,
    ROUND((COUNT(*) / (SELECT COUNT(*) FROM Usuarios WHERE eliminado = FALSE)) * 100, 2) AS PorcentajeDelTotal
FROM Usuarios u
JOIN CredencialAcceso c ON u.credencial_id = c.id
WHERE c.requiereReset = TRUE
    AND u.eliminado = FALSE
GROUP BY u.rol;
```

Contar usuarios por rol, y mostrar solo los roles con más de 3000 usuarios

```
SELECT rol, COUNT(*) AS cantidad
FROM Usuarios
WHERE eliminado = FALSE
GROUP BY rol
HAVING COUNT(*) > 3000;
```

# Bases de datos I

## Tiempo de ejecución de las consultas

Comando EXPLAIN ANALYZE

```
EXPLAIN ANALYZE SELECT * FROM view;  
1 *   SET @t1 = NOW(6);  
2      -- Reporte usuarios por rol y mes (GROUP BY + HAVING)  
3 *   SELECT  
4       DATE_FORMAT(u.fecha_registro, '%Y-%m') AS mes,  
5       u.rol,  
6       COUNT(*) AS total  
7   FROM Usuarios u  
8   WHERE u.eliminado = FALSE  
9   GROUP BY DATE_FORMAT(u.fecha_registro, '%Y-%m'), u.rol  
10  HAVING COUNT(*) >= 10  
11  ORDER BY mes DESC, total DESC LIMIT 10000000000000000000;  
12 *  SET @t2 = NOW(6);  
13 *  SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	duracion_microsegundos			
▶	207463			

Tiempos registrados

0.207463s

0.205046s

0.20486s

## Bases de datos I

```
1 *  SET @t1 = NOW(6);
2      -- Rango de usuarios registrados en fechas
3 *  SELECT id, username, email, fecha_registro
4      FROM Usuarios
5      WHERE fecha_registro BETWEEN '2025-10-20 19:27:50' AND '2025-10-20 19:49:04'
6          AND eliminado = FALSE LIMIT 10000000000000000000;
7 *  SET @t2 = NOW(6);
8 *  SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	duracion_microsegundos			
▶	170783			

Tiempos registrados

0.170783s

0.181968s

0.186269s

## Bases de datos I

```
1 * SET @t1 = NOW(6);
2   -- JOIN de usuarios con credencial
3 * SELECT u.id, u.username, u.rol, c.requiereReset, c.ultimoCambio
4   FROM Usuarios u
5   JOIN CredencialAcceso c ON u.credencial_id = c.id
6   WHERE c.requiereReset = TRUE AND u.activo = TRUE LIMIT 10000000000000000000;
7 * SET @t2 = NOW(6);
8 * SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
```

Result Grid	
<input type="button" value="Filter Rows"/>	<input type="button" value="Export"/>
	duracion_microsegundos
▶	61264

Tiempos registrados

0.061264s

0.065123s

0.065972s

## Bases de datos I

```
1 *  SET @t1 = NOW(6);
2      -- Subconsulta para verificar inconsistencia de datos
3 *
4     SELECT u.id, u.username
5     FROM Usuarios u
6     WHERE NOT EXISTS (
7         SELECT 1 FROM CredencialAcceso c WHERE c.id = u.credencial_id
8     ) AND u.eliminado = FALSE LIMIT 10000000000000000000;
9 *
10    SET @t2 = NOW(6);
11
12    SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	duracion_microsegundos			
▶	392822			

Tiempos registrados

0.392822s

0.394709s

0.393224s

# Bases de datos I

## Consultas de tiempo de ejecución con índice

```
1 *   SET @t1 = NOW(6);
2      -- Subconsulta para verificar inconsistencia de datos
3 *   SELECT
4     DATE_FORMAT(u.fecha_registro, '%Y-%m') AS mes,
5     u.rol,
6     COUNT(*) AS total
7   FROM Usuarios u FORCE INDEX (idx_rol, idx_fecha_registro)
8   WHERE u.eliminado = FALSE
9   GROUP BY DATE_FORMAT(u.fecha_registro, '%Y-%m'), u.rol
10  HAVING COUNT(*) >= 10
11  ORDER BY mes DESC, total DESC LIMIT 1000000000000000000;
12 *   SET @t2 = NOW(6);
13 *   SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
14
15
16
17
```

## Tiempos registrados

0.194867s

0.202841s

0.202858s

## Comando EXPLAIN ANALYZE

```
'-> Limit: 1000000000000000000 row(s) (actual time=203..203 rows=3 loops=1)\n  ->
Sort: mes DESC, total DESC (actual time=203..203 rows=3 loops=1)\n        -> Filter:
(`count(0)` >= 10) (actual time=203..203 rows=3 loops=1)\n              -> Table scan on
<temporary> (actual time=203..203 rows=3 loops=1)\n                  -> Aggregate using
temporary table (actual time=203..203 rows=3 loops=1)\n                      -> Filter:
(u.eliminado = false) (cost=86707 rows=24773) (actual time=0.0806..67.6 rows=248894
loops=1)\n              -> Table scan on u (cost=86707 rows=247734) (actual
time=0.0785..54.6 rows=248894 loops=1)\n'
```

## Bases de datos I

```
1 *   SET @t1 = NOW(6);
2 *   SELECT id, username, email, fecha_registro
3     FROM Usuarios FORCE INDEX (idx_fecha_registro)
4   WHERE fecha_registro BETWEEN '2025-01-01' AND '2025-06-30'
5     AND eliminado = FALSE LIMIT 10000000000000000;
6 *   SET @t2 = NOW(6);
7 *   SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
8
9
10
11
12
13
14
15
16 |
17
```

Tiempos registrados

0.016668s

0.014431s

0.016833s

Comando EXPLAIN ANALYZE

```
'-> Limit: 10000000000000000000 row(s) (cost=0.71 rows=0.1) (actual
time=0.0421..0.0421 rows=0 loops=1)\n  -> Filter: (usuarios.eliminado = false)
(cost=0.71 rows=0.1) (actual time=0.0412..0.0412 rows=0 loops=1)\n    -> Index
range scan on Usuarios using idx_fecha_registro over (\'2025-01-01 00:00:00\'\'
<= fecha_registro <= \'2025-06-30 00:00:00\'), with index condition:
(usuarios.fecha_registro between \'2025-01-01\' and \'2025-06-30\')
(cost=0.71 rows=1) (actual time=0.0406..0.0406 rows=0 loops=1)\n'
```

# Bases de datos I

```
1 *   SET @t1 = NOW();
2 *   SELECT u.id, u.username, u.rol, c.requiereReset, c.ultimoCambio
3     FROM Usuarios u FORCE INDEX (idx_credencial_id)
4       JOIN CredencialAcceso c FORCE INDEX (idx_requiereReset)
5         ON u.credencial_id = c.id
6       WHERE c.requiereReset = TRUE AND u.activo = TRUE LIMIT 10000000000000000000;
7 *   SET @t2 = NOW();
8 *   SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
9
10
11
12
13
14
15
16
17
```

Tiempos registrados

0.01264s

0.011273s

0.014147s

Comando EXPLAIN ANALYZE

```
'-> Limit: 10000000000000000000 row(s) (cost=11.9 rows=1.7) (actual
time=0.0714..0.244 rows=19 loops=1)\n  -> Nested loop inner join (cost=11.9
rows=1.7) (actual time=0.0706..0.241 rows=19 loops=1)\n    -> Index lookup on
c using idx_requiereReset (requiereReset=true) (cost=5.95 rows=17) (actual
time=0.0533..0.108 rows=17 loops=1)\n      -> Filter: (u.activo = true)
(cost=0.251 rows=0.1) (actual time=0.00596..0.00739 rows=1.12 loops=17)\n
-> Index lookup on u using idx_credencial_id (credencial_id=c.id) (cost=0.251
rows=1) (actual time=0.00567..0.00701 rows=1.12 loops=17)\n'
```

## Bases de datos I

```
1 *   SET @t1 = NOW(6);
2 *   SELECT u.id, u.username
3     FROM Usuarios u FORCE INDEX (idx_credencial_id)
4   WHERE NOT EXISTS (
5       SELECT 1 FROM CredencialAcceso c FORCE INDEX (PRIMARY)
6         WHERE c.id = u.credencial_id
7   )
8     AND u.eliminado = FALSE LIMIT 10000000000000000000;
9 *
10 *  SET @t2 = NOW(6);
11
12
13
14 |
15
16
17
```

Tiempos registrados

0.415924s

0.394796s

0.404986s

Comando EXPLAIN ANALYZE

```
'-> Limit: 10000000000000000000 row(s) (cost=95378 rows=24773) (actual
time=406..406 rows=0 loops=1)\n  -> Nested loop antijoin (cost=95378
rows=24773) (actual time=406..406 rows=0 loops=1)\n    -> Filter: (u.eliminado
= false) (cost=86707 rows=24773) (actual time=0.0546..73.6 rows=248894
loops=1)\n      -> Table scan on u (cost=86707 rows=247734) (actual
time=0.0536..60.7 rows=248894 loops=1)\n      -> Single-row covering index
lookup on c using PRIMARY (id=u.credencial_id) (cost=0.25 rows=1) (actual
time=0.00128..0.00128 rows=1 loops=248894)\n'
```

# Bases de datos I

```
1 *   SET @t1 = NOW(6);
2 *   SELECT * FROM vw_active_users_cred LIMIT 10000000000000000000;
3 *   SET @t2 = NOW(6);
4 *   SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
5
6
7
8
9
10
11
12
13
14 |
15
16
17
```

Tiempos registrados

1.511211s

1.349046s

1.397063s

Comando EXPLAIN ANALYZE

```
'-> Limit: 10000000000000000000 row(s) (cost=170461..170764 rows=24009)
(actual time=861..890 rows=248894 loops=1)\n  -> Table scan on
vw_active_users_cred (cost=170461..170764 rows=24009) (actual
time=861..881 rows=248894 loops=1)\n        -> Materialize (cost=170461..170461
rows=24009) (actual time=861..861 rows=248894 loops=1)\n              -> Limit:
10000000000000000000 row(s) (cost=168060 rows=24009) (actual
time=0.0702..0.725 rows=248894 loops=1)\n                  -> Nested loop inner join
(cost=168060 rows=24009) (actual time=0.0695..0.713 rows=248894 loops=1)\n
-> Table scan on c (cost=84030 rows=240086) (actual time=0.0436..56.3
rows=248892 loops=1)\n                  -> Filter: (u.eliminado = false) (cost=0.25
rows=0.1) (actual time=0.00214..0.00251 rows=1 loops=248892)\n
-> Index lookup on u using idx_credencial_id (credencial_id=c.id) (cost=0.25
rows=1) (actual time=0.00205..0.00238 rows=1 loops=248892)\n'
```

# Bases de datos I

## Etapa 4. Seguridad e integridad

### Creación de usuarios

```
1      -- 1. Usuario Operacional de la Aplicación (CRUD)
2 *  CREATE USER 'app_user'@'localhost' IDENTIFIED BY 'w8THiLn7N383';
3 *  GRANT SELECT, INSERT, UPDATE, DELETE ON credencialesusuarios.Usuarios TO 'app_user'@'localhost';
4 *  GRANT SELECT, INSERT, UPDATE, DELETE ON credencialesusuarios.CredencialAcceso TO 'app_user'@'localhost';
5
6      -- 2. Usuario de Reportes (Solo lectura y sin acceso a hashes)
7 *  CREATE USER 'app_report'@'localhost' IDENTIFIED BY '1J73Ykfa8JR';
8      -- Le damos SELECT solo sobre la vista, no sobre las tablas directamente.
9 *  GRANT SELECT ON credencialesusuarios.ReporteSeguridad TO 'app_report'@'localhost';
10
11 *  FLUSH PRIVILEGES;
```

### Vista reporte seguridad

	id	username	email	eliminado	activo	fecha_registro	rol	credencial_id	ultimoCambio	requiereReset
1	Lauti	zulofolu@gmail.com	0	1	2025-10-20 19:27:50	Administrador	1	2025-10-20 20:27:32	1	
4	usuario_1	usuario1@correo.com	0	1	2025-10-20 19:49:04	Usuario	1	2025-10-20 20:27:32	1	
2	Walter	walter@gmail.com	0	1	2025-10-20 19:28:22	Moderador	2	2025-10-20 20:27:32	1	
5	usuario_2	usuario2@correo.com	0	1	2025-10-20 19:49:04	Usuario	2	2025-10-20 20:27:32	1	
6	usuario_3	usuario3@correo.com	0	1	2025-10-20 19:49:04	Usuario	3	2025-10-20 20:27:32	1	
7	usuario_4	usuario4@correo.com	0	1	2025-10-20 19:49:04	Usuario	4	2025-10-20 20:27:32	1	
8	usuario_5	usuario5@correo.com	0	1	2025-10-20 19:49:04	Usuario	5	2025-10-20 20:27:32	1	
9	usuario_6	usuario6@correo.com	0	1	2025-10-20 19:49:04	Usuario	6	2025-10-20 19:43:33	0	
10	usuario_7	usuario7@correo.com	0	1	2025-10-20 19:49:04	Usuario	7	2025-10-20 19:43:33	0	
11	usuario_8	usuario8@correo.com	0	1	2025-10-20 19:49:04	Usuario	8	2025-10-20 19:43:33	0	
12	usuario_9	usuario9@correo.com	0	1	2025-10-20 19:49:04	Usuario	9	2025-10-20 19:43:33	0	
13	usuario_10	usuario10@correo.com	0	1	2025-10-20 19:49:04	Moderador	10	2025-10-20 19:43:33	0	
14	usuario_11	usuario11@correo.com	0	1	2025-10-20 19:49:04	Usuario	11	2025-10-20 19:43:33	0	
15	usuario_12	usuario12@correo.com	0	1	2025-10-20 19:49:04	Usuario	12	2025-10-20 19:43:33	0	
16	usuario_13	usuario13@correo.com	0	1	2025-10-20 19:49:04	Usuario	13	2025-10-20 19:43:33	0	
17	usuario_14	usuario14@correo.com	0	1	2025-10-20 19:49:04	Usuario	14	2025-10-20 20:22:22	1	

# Bases de datos I

## Creación de procedimiento

```
1      DELIMITER $$  
2  
3  ●  CREATE PROCEDURE sp_GetReporteSeguridad(  
4          IN p_rol VARCHAR(20),  
5          IN p_activo TINYINT,  
6          IN p_fecha_inicio DATETIME,  
7          IN p_fecha_fin DATETIME  
8      )  
9      BEGIN  
10         SELECT  
11             u.id,  
12             u.username,  
13             u.email,  
14             u.eliminado AS usuario_eliminado,  
15             u.activo AS usuario_activo,  
16             u.fecha_registro,  
17             u.rol,  
18             u.credencial_id,  
19             c.ultimoCambio,  
20             c.requiereReset  
21         FROM Usuarios u  
22         INNER JOIN CredencialAcceso c ON u.credencial_id = c.id  
23         WHERE c.eliminado = FALSE  
24             AND (p_rol IS NULL OR u.rol = p_rol)  
25             AND (p_activo IS NULL OR u.activo = p_activo)  
26             AND (p_fecha_inicio IS NULL OR u.fecha_registro >= p_fecha_inicio)  
27             AND (p_fecha_fin IS NULL OR u.fecha_registro <= p_fecha_fin)  
28         ORDER BY u.fecha_registro DESC;  
29     END$$  
30  
31     DELIMITER ;  
32  
33  ●  CALL sp_GetReporteSeguridad(NULL, NULL, NULL, NULL);  
34  
35  ●  CALL sp_GetReporteSeguridad(NULL, NULL, "2025-01-01"; DROP TABLE Usuarios; --", NULL);
```

Comenzamos creando un procedimiento llamado “sp\_GetReporteSeguridad”, dentro del primer paréntesis. Tenemos los parámetros de nuestro procedimiento donde vamos a establecer qué datos se pueden pasar y de qué tipo de valores deben ser esos datos.

Luego llegamos al BEGIN donde va a estar el cuerpo de nuestro procedimiento, dentro de este bloque de código vamos a poner nuestra sentencia SQL. En este caso lo que queremos que haga nuestro procedimiento es obtener todos los datos no sensibles de los usuarios filtrando por rol, activo y rango de fecha de creación. Hacemos una consulta con un JOIN para obtener todos los datos necesarios.

## Bases de datos I

```
SELECT
    u.id,
    u.username,
    u.email,
    u.eliminado AS usuario_eliminado,
    u.activo AS usuario_activo,
    u.fecha_registro,
    u.rol,
    u.credencial_id,
    c.ultimoCambio,
    c.requiereReset
FROM Usuarios u
INNER JOIN CredencialAcceso c ON u.credencial_id = c.id
```

Y al final un WHERE para poder filtrar por los datos ingresados en la consulta.

```
WHERE c.eliminado = FALSE
AND (p_rol IS NULL OR u.rol = p_rol)
AND (p_activo IS NULL OR u.activo = p_activo)
AND (p_fecha_inicio IS NULL OR u.fecha_registro >= p_fecha_inicio)
AND (p_fecha_fin IS NULL OR u.fecha_registro <= p_fecha_fin)
ORDER BY u.fecha_registro DESC;
```

Dentro de cada paréntesis vamos a verificar si el filtro ingresado por el usuario que ejecuta la consulta es distinto de NULL, en ese caso vamos a hacer la correspondiente comparación para obtener los datos deseados. Caso contrario se ignora el filtro.

Caso correcto

33 • CALL sp\_GetReporteSeguridad(NULL, NULL, NULL, NULL);

	id	username	email	usuario_eliminado	usuario_activo	fecha_registro	rol	credencial_id	ultimoCambio	requiereReset
▶	248643	usuario_248637	usuario248637@correo.com	0	1	2025-10-20 19:52:22	Usuario	248637	2025-10-20 19:48:31	0
	248644	usuario_248638	usuario248638@correo.com	0	1	2025-10-20 19:52:22	Usuario	248638	2025-10-20 19:48:31	0
	248645	usuario_248639	usuario248639@correo.com	0	1	2025-10-20 19:52:22	Usuario	248639	2025-10-20 19:48:31	0
	248646	usuario_248640	usuario248640@correo.com	0	1	2025-10-20 19:52:22	Moderador	248640	2025-10-20 19:48:31	0
	248647	usuario_248641	usuario248641@correo.com	0	1	2025-10-20 19:52:22	Usuario	248641	2025-10-20 19:48:31	0
	248648	usuario_248642	usuario248642@correo.com	0	1	2025-10-20 19:52:22	Usuario	248642	2025-10-20 19:48:31	0
	248649	usuario_248643	usuario248643@correo.com	0	1	2025-10-20 19:52:22	Usuario	248643	2025-10-20 19:48:31	0

# Bases de datos I

## Caso incorrecto

The screenshot shows a database interface with the following details:

- SQL Statement:** 34 \* CALL sp\_GetReporteSeguridad(NULL, NULL, "2025-01-01'; DROP TABLE Usuarios; --", NULL);
- Output Tab:** Action Output
- Log Row:** 1 19:49:16 CALL sp\_GetReporteSeguridad(NULL, NULL, "2025-01-01'; DROP TABLE Usuarios; --", NULL)
- Message:** Error Code: 1292. Incorrect datetime value: '2025-01-01'; DROP TABLE Usuarios; --' for column 'p\_fecha\_inicio' at row 1

# Bases de datos I

## Etapa 5. Conurrencia y transacciones

### Deadlock

#### consola 1

The screenshot shows a MySQL Workbench interface. The top part is a query editor window with the following SQL code:

```
1 • START TRANSACTION;
2 • UPDATE usuarios SET activo = 0 WHERE id = 1;
3
4 • UPDATE credencialacceso SET eliminado = 0 WHERE id = 1;
```

The bottom part is an "Output" window titled "Action Output". It lists the execution of the four statements with their times and messages:

#	Time	Action	Message
6	19:51:44	UPDATE usuarios SET activo = 0 WHERE id = 1	0 row(s) affected Rows matched: 1 Changed: 0 Warnings: 0
7	19:51:53	SELECT * FROM credencialesusuarios.usuarios LIMIT 0, 1000	1000 row(s) returned
8	19:52:03	SELECT * FROM credencialesusuarios.credencialacceso LIMIT 0, 1000	1000 row(s) returned
9	19:52:33	SELECT * FROM credencialesusuarios.credencialacceso LIMIT 0, 1000	1000 row(s) returned
10	19:54:26	UPDATE credencialacceso SET eliminado = 0 WHERE id = 1	Error Code: 1213. Deadlock found when trying to get lock; try restarting transaction

#### consola 2

The screenshot shows a MySQL Workbench interface. The top part is a query editor window with the following SQL code:

```
1 • START TRANSACTION;
2 • UPDATE credencialacceso SET eliminado = 1 WHERE id = 1;
3
4 • UPDATE usuarios SET activo = 1 WHERE id = 1;
```

# Bases de datos I

## Deadlock con rollback

The screenshot shows the MySQL Workbench interface. In the top tab bar, the active tab is 'SQL File 11\*'. Below the tabs, there's a toolbar with icons for file operations, search, and refresh. A status bar at the bottom indicates 'Limit to 1000 rows'.

The SQL editor contains the following code:

```
41      -- Intento de transacción
42      START TRANSACTION;
43      UPDATE Usuarios
44          SET activo = p_nuevo_estado
45          WHERE id = p_user_id;
46      UPDATE CredencialAcceso
47          SET ultimoCambio = NOW()
48          WHERE id = (SELECT credencial_id FROM Usuarios WHERE id = p_user_id);
49      COMMIT;
50      SELECT 'Transacción completada exitosamente.' AS mensaje;
51      SET v_done = 1;
52  END;
53  END WHILE;
54 END //
55 DELIMITER ;
56
57
58 • START TRANSACTION;
59 • CALL actualizar_estado_usuario_retry(1, 0);
60
61 • CALL actualizar_estado_usuario_retry(1, 0);
```

The 'Result Grid' pane shows a single row of results:

mensaje
X: Lock timeout detected. Reintento número 1

The 'Output' pane shows the execution log:

#	Time	Action	Message
1	20:45:41	START TRANSACTION	0 row(s) affected
2	20:45:49	CALL actualizar_estado_usuario_retry(1, 0)	1 row(s) returned

The screenshot shows the MySQL Workbench interface. The active tab is 'SQL File 11\*'. The SQL editor contains the same code as the previous screenshot.

The 'Result Grid' pane shows a single row of results:

mensaje
Error persistente: se canceló la transacción tras múltiples intentos.

The screenshot shows the MySQL Workbench interface. The active tab is 'SQL File 11\*'. The SQL editor contains the same code as the previous screenshots.

The 'Result Grid' pane shows a single row of results:

mensaje
Transacción completada exitosamente.

The 'Output' pane shows the execution log:

#	Time	Action	Message
1	20:45:41	START TRANSACTION	0 row(s) affected
2	20:45:49	CALL actualizar_estado_usuario_retry(1, 0)	1 row(s) returned
3	20:45:55	CALL actualizar_estado_usuario_retry(1, 0)	1 row(s) returned
4	20:45:59	CALL actualizar_estado_usuario_retry(1, 0)	1 row(s) returned
5	20:45:59	CALL actualizar_estado_usuario_retry(1, 0)	1 row(s) returned

Observación: Tuvimos que agregar un timeout de 3s sino la conexión con sql se perdía.

# Bases de datos I

## READ COMMITTED

```
1 SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
2 START TRANSACTION;
3 SELECT username, email FROM Usuarios WHERE id = 1;
4
5 SELECT username, email FROM Usuarios WHERE id = 1;
```

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

username	email
Lauti	nuevo@email.com

Usuarios 2 ×

Output

#	Time	Action	Message
10	19:54:26	UPDATE credencialacceso SET eliminado = 0 WHERE id = 1	Error Code: 1213. Deadlock found when trying to get lock; try restarting transaction
11	20:02:38	SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED	0 row(s) affected
12	20:02:38	START TRANSACTION	0 row(s) affected
13	20:02:38	SELECT username, email FROM Usuarios WHERE id = 1 LIMIT 0, 1000	1 row(s) returned
14	20:03:45	SELECT username, email FROM Usuarios WHERE id = 1 LIMIT 0, 1000	1 row(s) returned

## REPEATABLE READ

```
1 SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
2 START TRANSACTION;
3
4 SELECT username, email FROM Usuarios WHERE id = 1;
5
6 SELECT username, email FROM Usuarios WHERE id = 1;
```

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

username	email
Lauti	repetible@email.com

Usuarios 4 ×

Como podemos observar la diferencia entre read committed y repeatable read es que read committed permite el uso o cambio en la tabla que se está utilizando y muestra los cambios en vivo mientras que repeatable read bloquea esta tabla hasta que la transacción se haya completado.

Esto nos permitirá evitar errores de deadlock ya que este error significa que 2 transacciones se bloquean por ejecutarse simultáneamente utilizando los

# Bases de datos I

mismos datos.

## Conclusión

Gracias a este trabajo práctico pudimos aprender cómo sería la creación y seteo de una base de datos lo más cercano a lo que sería un entorno de producción, pudiendo aplicar conceptos como recurrencia que es importante para cuando se ejecutan muchas consultas en simultáneo en nuestro backend, seguridad para protegernos contra ataques de sql injection y consultas complejas para obtener datos más detallados. También pudimos observar la mejora en eficiencia de las consultas aplicando índices y pudiendo llevar este aprendizaje a nuestra carrera profesional.

## Video

<https://youtu.be/Qtiv46I-AKE>