Universidad Tecnológica Nacional

Bases de datos I

Grupo:
- Lautaro Zullo
- Walter Verdún
- Ignacio Salazar

Tutor: Constanza Uño

Profesor: Cinthia Rigoni

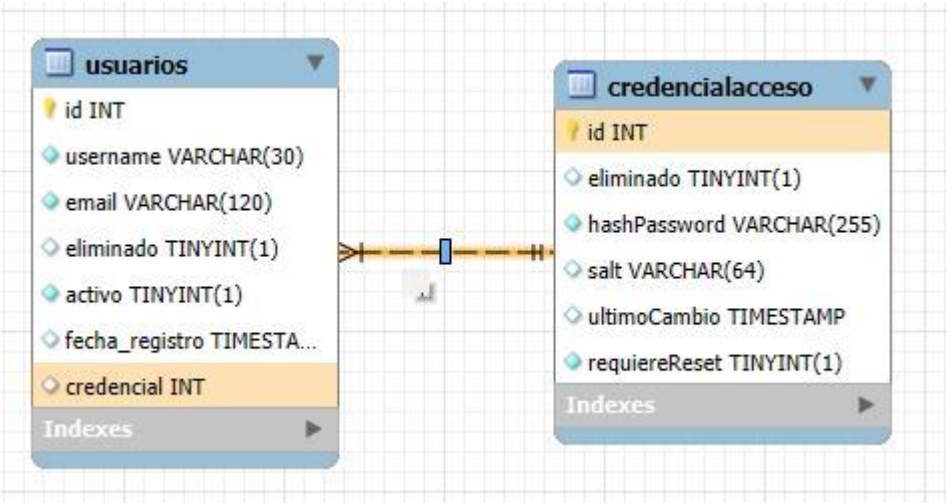
23 de octubre, 2025

ÍNDICE

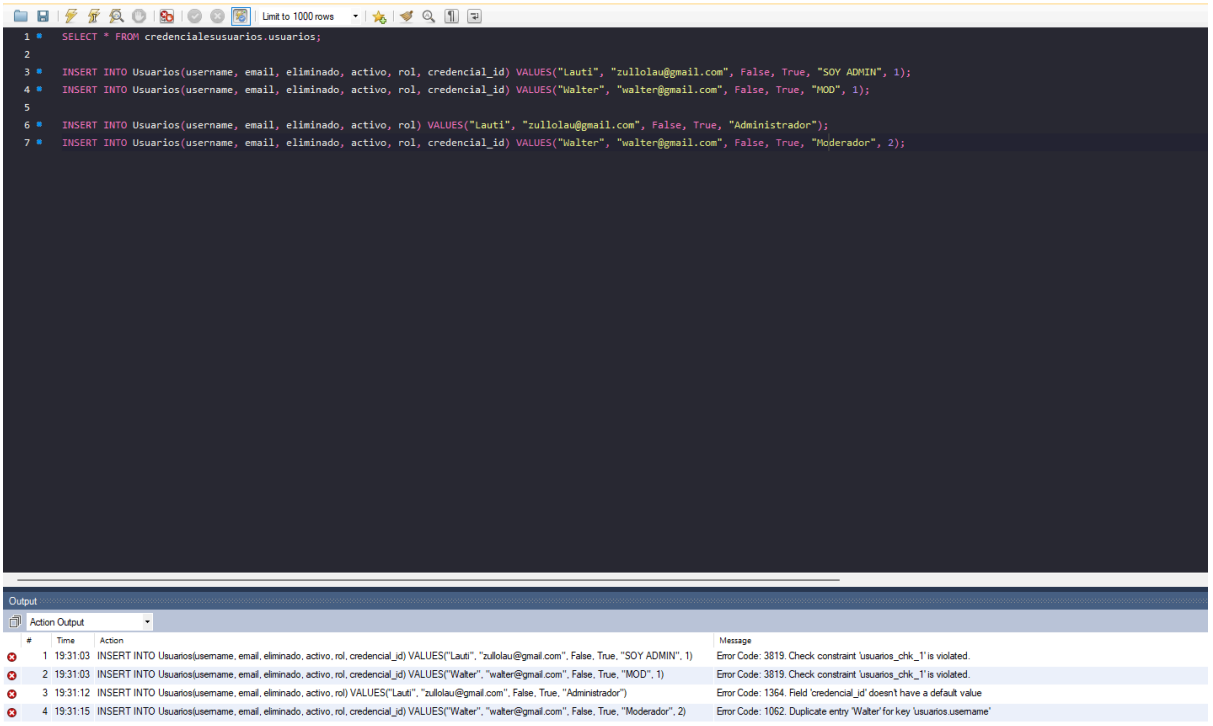
Resolución de consignas

Etapa 1 – Modelado y Definición de Constraints

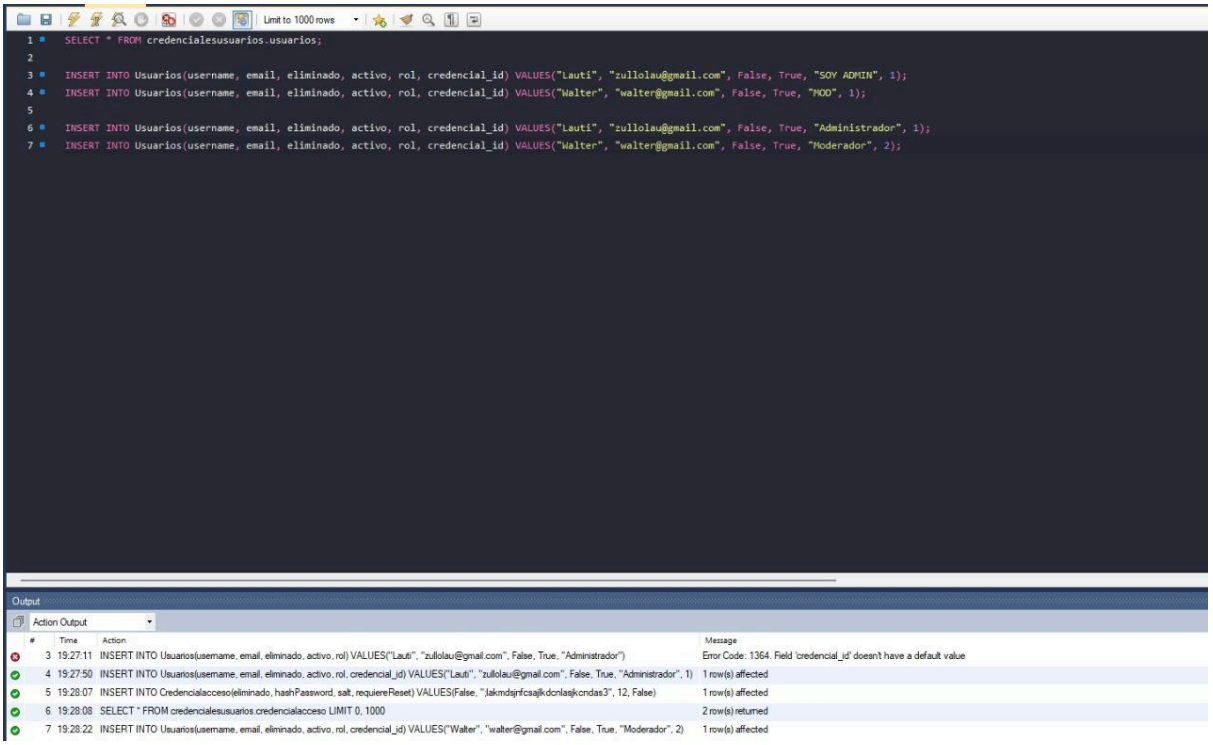
Diagrama EER



Inserción de 2 filas erróneas



Inserción de 2 filas validadas



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);
```

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);
```

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);
```

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

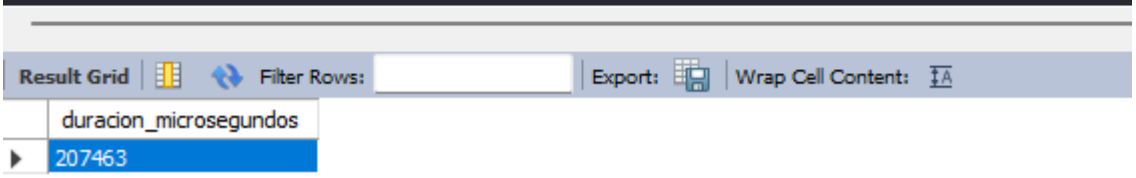
```
SELECT rol, COUNT(*) AS cantidad
FROM Usuarios
WHERE eliminado = FALSE
GROUP BY rol
HAVING COUNT(*) > 3000;
```

Tiempo de ejecución de las consultas

Comando EXPLAIN ANALYZE

```
EXPLAIN ANALYZE SELECT * FROM view;
```

```
SET @t1 = NOW(6);
-- Reporte usuarios por rol y mes (GROUP BY + HAVING)
SELECT
  DATE_FORMAT(u.fecha_registro, '%Y-%m') AS mes,
  u.rol,
  COUNT(*) AS total
FROM Usuarios u
WHERE u.eliminado = FALSE
GROUP BY DATE_FORMAT(u.fecha_registro, '%Y-%m'), u.rol
HAVING COUNT(*) >= 10
ORDER BY mes DESC, total DESC LIMIT 100000000000000000;
SET @t2 = NOW(6);
SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
```



-

Tiempos registrados

0.207463s
0.205046s
0.20486s

```sql
SET @t1 = NOW(6);
-- Rango de usuarios registrados en fechas
SELECT id, username, email, fecha_registro
FROM Usuarios
WHERE fecha_registro BETWEEN '2025-10-20 19:27:50' AND '2025-10-20 19:49:04'
  AND eliminado = FALSE LIMIT 10000000000000000;
SET @t2 = NOW(6);
SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
```

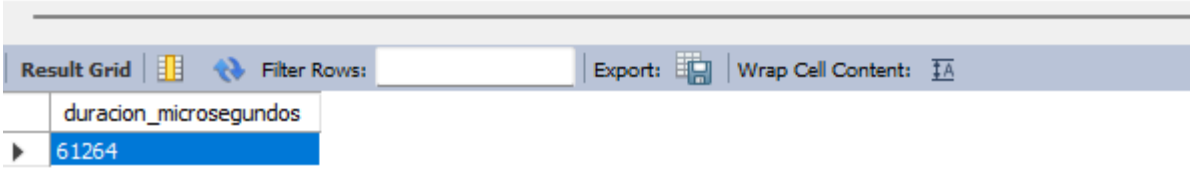
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

duracion_microsegundos
170783

Tiempos registrados

0.170783s
0.181968s
0.186269s

```sql
SET @t1 = NOW(6);
-- JOIN de usuarios con credencial
SELECT u.id, u.username, u.rol, c.requiereReset, c.ultimoCambio
FROM Usuarios u
JOIN CredencialAcceso c ON u.credencial_id = c.id
WHERE c.requiereReset = TRUE AND u.activo = TRUE LIMIT 100000000000000000;
SET @t2 = NOW(6);
SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
```



duracion_microsegundos
61264

Tiempos registrados

0.061264s
0.065123s
0.065972s

```sql
SET @t1 = NOW(6);
-- Subconsulta para verificar inconsistencia de datos
SELECT u.id, u.username
FROM Usuarios u
WHERE NOT EXISTS (
  SELECT 1 FROM CredencialAcceso c WHERE c.id = u.credencial_id
) AND u.eliminado = FALSE LIMIT 10000000000000000;
SET @t2 = NOW(6);
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

Consultas de tiempo de ejecución con índice

```sql
SET @t1 = NOW(6);
-- Subconsulta para verificar inconsistencia de datos
SELECT
  DATE_FORMAT(u.fecha_registro, '%Y-%m') AS mes,
  u.rol,
  COUNT(*) AS total
FROM Usuarios u FORCE INDEX (idx_rol, idx_fecha_registro)
WHERE u.eliminado = FALSE
GROUP BY DATE_FORMAT(u.fecha_registro, '%Y-%m'), u.rol
HAVING COUNT(*) >= 10
ORDER BY mes DESC, total DESC LIMIT 10000000000000000;
SET @t2 = NOW(6);
SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
```

Tiempos registrados

0.194867s
0.202841s
0.202858s

Comando EXPLAIN ANALYZE

'-> Limit: 10000000000000000 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'

```sql
SET @t1 = NOW(6);
SELECT id, username, email, fecha_registro
FROM Usuarios FORCE INDEX (idx_fecha_registro)
WHERE fecha_registro BETWEEN '2025-01-01' AND '2025-06-30'
  AND eliminado = FALSE LIMIT 10000000000000000;
SET @t2 = NOW(6);
SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
```

Tiempos registrados

0.016668s
0.014431s
0.016833s

Comando EXPLAIN ANALYZE

'-> Limit: 100000000000000000 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'

```sql
SET @t1 = NOW(6);
SELECT u.id, u.username, u.rol, c.requiereReset, c.ultimoCambio
FROM Usuarios u FORCE INDEX (idx_credencial_id)
JOIN CredencialAcceso c FORCE INDEX (idx_requiereReset)
  ON u.credencial_id = c.id
WHERE c.requiereReset = TRUE AND u.activo = TRUE LIMIT 100000000000000000;
SET @t2 = NOW(6);
SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
```

Tiempos registrados

0.01264s
0.011273s
0.014147s

Comando EXPLAIN ANALYZE

'-> Limit: 100000000000000000 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'

```sql
SET @t1 = NOW(6);
SELECT u.id, u.username
FROM Usuarios u FORCE INDEX (idx_credencial_id)
WHERE NOT EXISTS (
  SELECT 1 FROM CredencialAcceso c FORCE INDEX (PRIMARY)
  WHERE c.id = u.credencial_id
)
AND u.eliminado = FALSE LIMIT 10000000000000000;
SET @t2 = NOW(6);
SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
```

Tiempos registrados

0.415924s
0.394796s
0.404986s

Comando EXPLAIN ANALYZE

'-> Limit: 100000000000000000 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'

```sql
SET @t1 = NOW(6);
SELECT * FROM vw_active_users_cred LIMIT 100000000000000000;
SET @t2 = NOW(6);
SELECT TIMESTAMPDIFF(MICROSECOND, @t1, @t2) AS duracion_microsegundos;
```

Tiempos registrados

1.511211s
1.349046s
1.397063s

Comando EXPLAIN ANALYZE

'-> Limit: 100000000000000000 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: 100000000000000000 row(s) (cost=168060 rows=24009) (actual time=0.0702..725 rows=248894 loops=1)\n   -> Nested loop inner join (cost=168060 rows=24009) (actual time=0.0695..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'

Etapa 4. Seguridad e integridad

Creación de usuarios

```sql
-- 1. Usuario Operacional de la Aplicación (CRUD)
CREATE USER 'app_user'@'localhost' IDENTIFIED BY 'w8THiLn7N383';
GRANT SELECT, INSERT, UPDATE, DELETE ON credencialesusuarios.Usuarios TO 'app_user'@'localhost';
GRANT SELECT, INSERT, UPDATE, DELETE ON credencialesusuarios.CredencialAcceso TO 'app_user'@'localhost';

-- 2. Usuario de Reportes (Solo lectura y sin acceso a hashes)
CREATE USER 'app_report'@'localhost' IDENTIFIED BY '1J73Ykfa8JRs';
-- Le damos SELECT solo sobre la vista, no sobre las tablas directamente.
GRANT SELECT ON credencialesusuarios.ReporteSeguridad TO 'app_report'@'localhost';

FLUSH PRIVILEGES;
```

Vista reporte seguridad

```sql
CREATE OR REPLACE VIEW ReporteSeguridad AS SELECT u.*, c.ultimoCambio, c.requiereReset FROM Usuarios u INNER JOIN CredencialAcceso c ON u.credencial_id = c.id WHERE c.eliminado = FALSE;

SELECT * FROM ReporteSeguridad;
```

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

id	username	email	eliminado	activo	fecha_registro	rol	credencial_id	ultimoCambio	requiereReset
1	Lauti	zullolau@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

ReporteSeguridad 2

Creación de procedimiento

```sql
DELIMITER $$

CREATE PROCEDURE sp_GetReporteSeguridad(
    IN p_rol VARCHAR(20),
    IN p_activo TINYINT,
    IN p_fecha_inicio DATETIME,
    IN p_fecha_fin DATETIME
)
BEGIN
    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
    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;
END$$

DELIMITER ;

CALL sp_GetReporteSeguridad(NULL, NULL, NULL, NULL);

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.

```
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

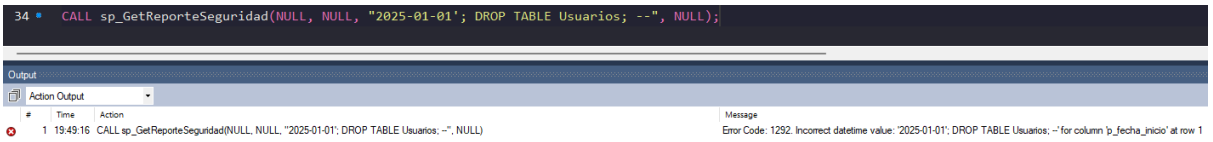
```
33 •  CALL sp_GetReporteSeguridad(NULL, NULL, NULL, NULL);
```

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

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

Result 5

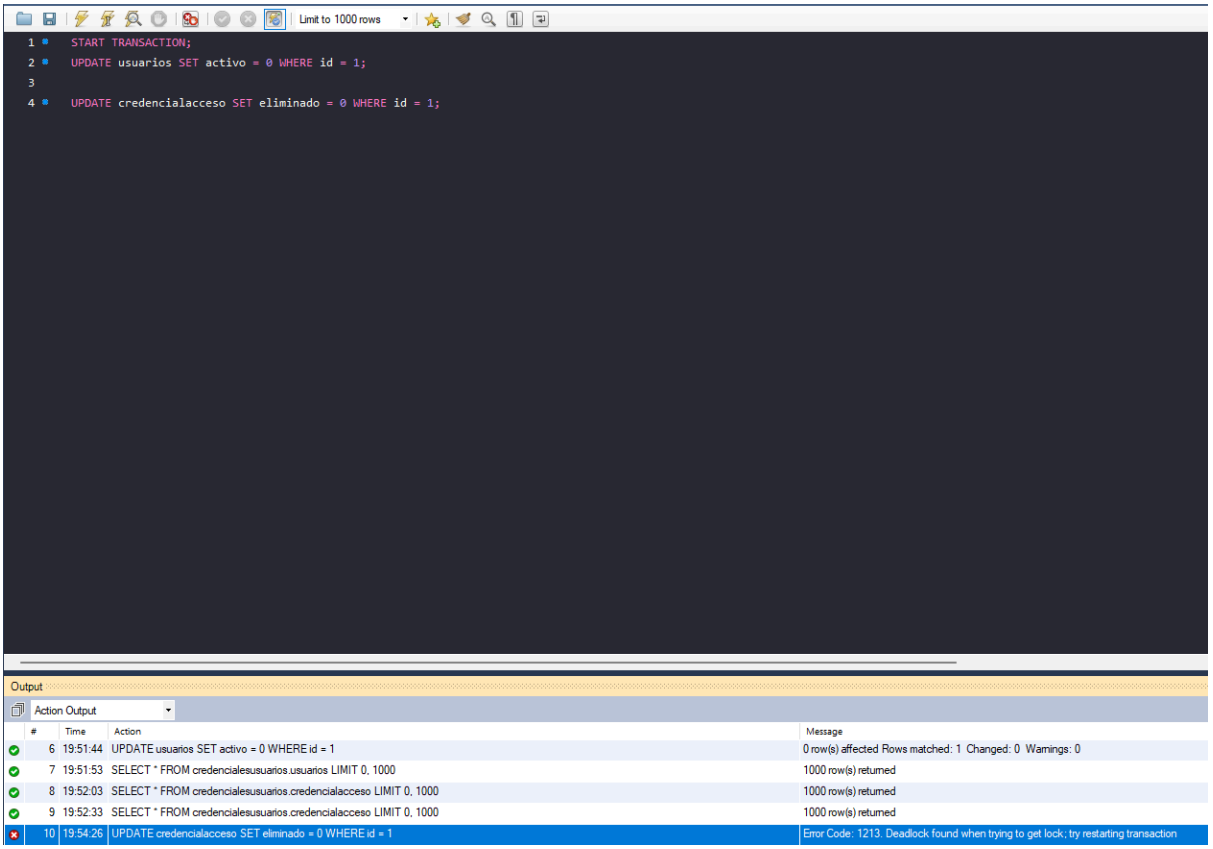
Caso incorrecto



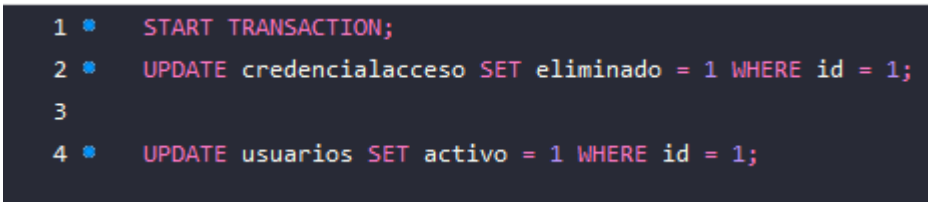
Etapa 5. Concurrencia y transacciones

Deadlock

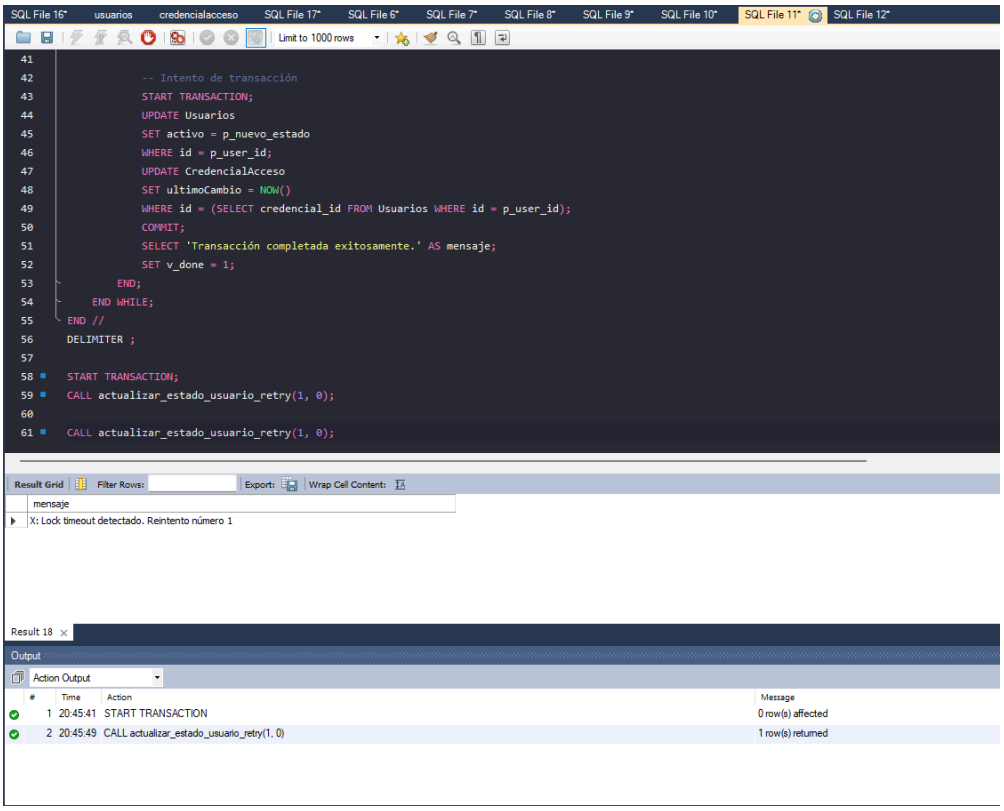
consola 1

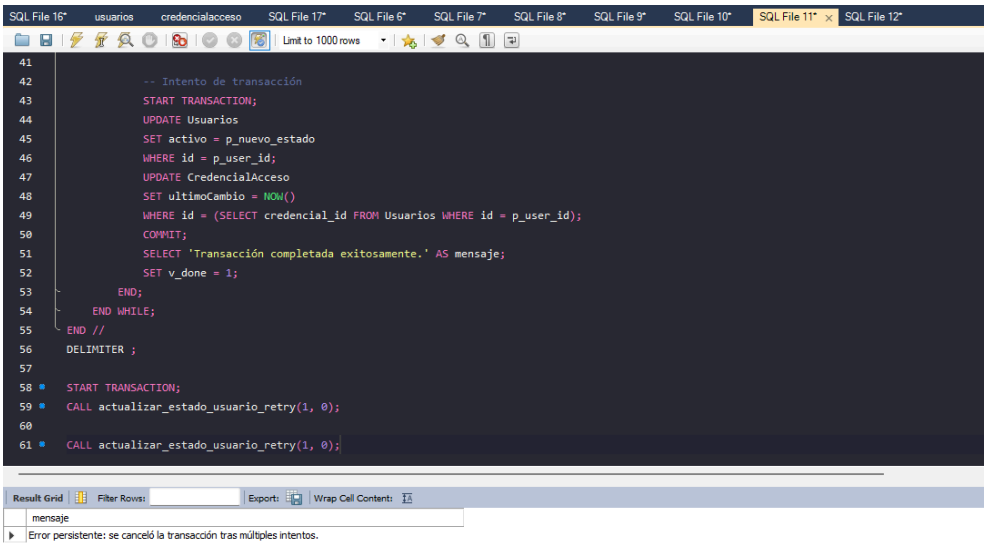


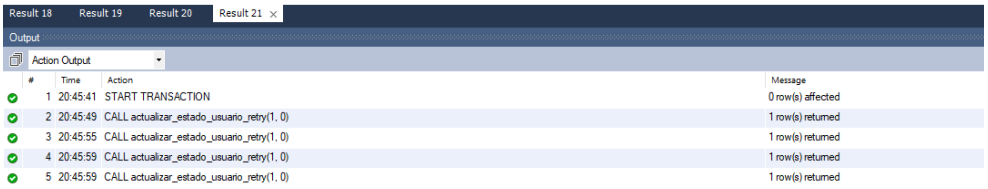
consola 2



Deadlock con rollback

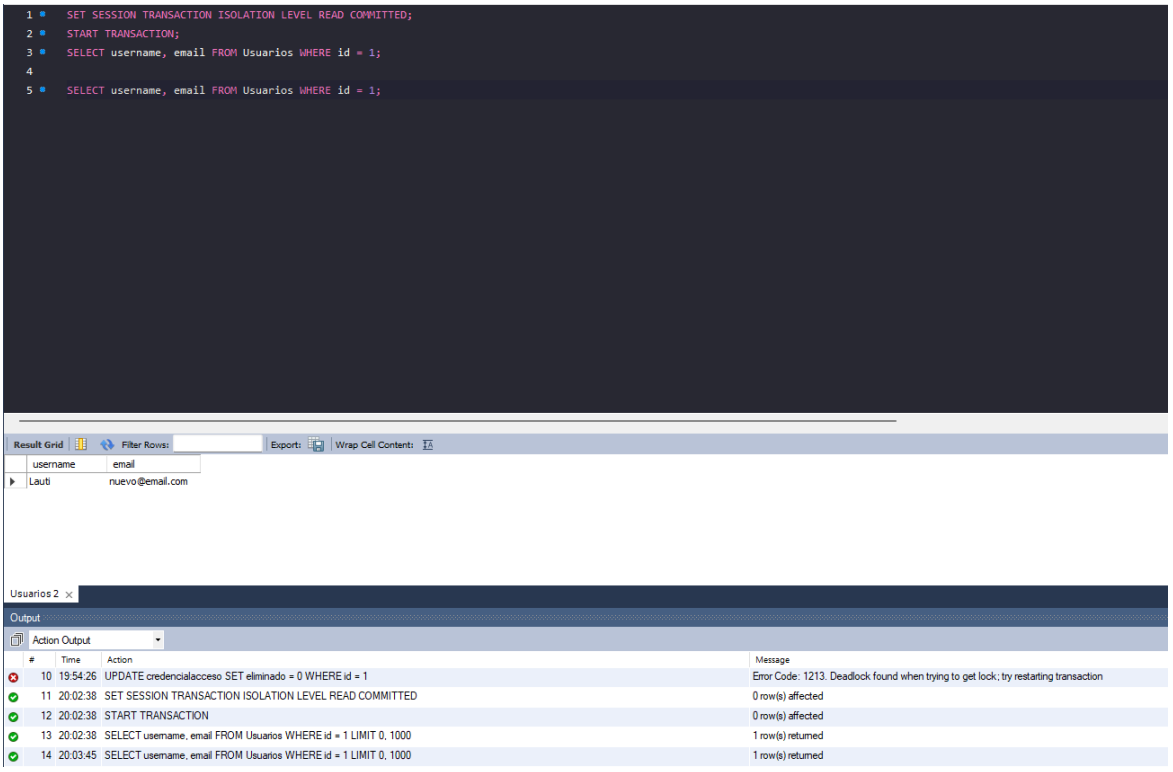




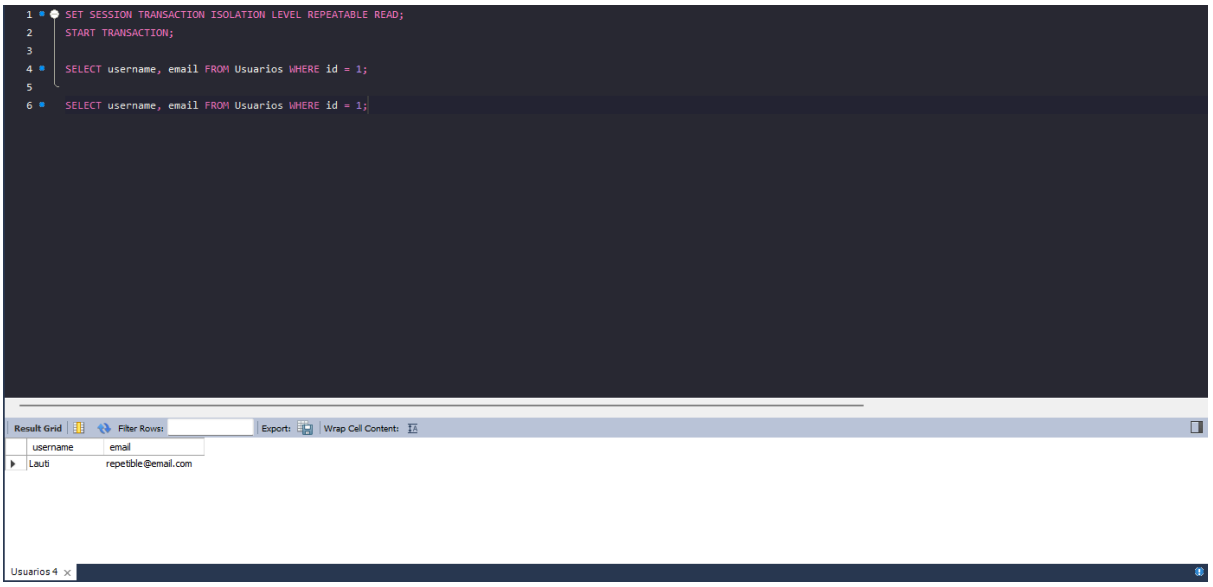


Observación: Tuvimos que agregar un timeout de 3s sino la conexión con sql se perdía.

READ COMMITTED



REPEATABLE READ



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

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