Lab8 - Transacciones

Pepe Calderón, Fernando Sola, Daniel Ayala, Inma Hernández, Margarita Cruz, Carlos Arévalo, David Ruiz

Escuela Técnica Superior de
Ingeniería Informática

Índice

1 Objetivo

El objetivo de esta práctica es implementar transacciones en SQL. El alumno aprenderá a:

- Comprender el concepto de transacción como Unidad Lógica de Trabajo.
- Activar/desactivar el `AUTOCOMMIT`.
- Usar `START TRANSACTION`, `COMMIT` y `ROLLBACK` para controlar transacciones.
- Implementar control de transacciones en procedimientos almacenados.
- Manejar excepciones con `DECLARE EXIT HANDLER`.
- Diferenciar entre procedimientos transaccionales y no transaccionales.
- Observar el comportamiento de transacciones concurrentes.

2 Preparación del entorno

Abre HeidiSQL y conéctate con el usuario `iissi_user` a la base de datos `GradesDB`. Asegúrate de haber ejecutado previamente los scripts `createDB.sql` y `populateDB.sql`.

3 Control de versiones

Continuaremos trabajando con el repositorio `GradesDB` creado en L1.

Al inicio del laboratorio, crea el archivo de transacciones:

```
git add transactions.sql
git commit -m "Añadido archivo transactions.sql para L8"
```

Al finalizar el laboratorio, haz push al repositorio remoto:

```
git add transactions.sql
git commit -m "Completado L8: Transacciones SQL"
git push origin main
```

4

Concepto de transacción

Una **transacción** es una Unidad Lógica de Trabajo (ULT) que agrupa un conjunto de operaciones que deben ejecutarse de forma **atómica**: o se ejecutan todas correctamente, o ninguna tiene efecto.

4.1. Propiedades ACID

Las transacciones deben cumplir las propiedades ACID:

- **Atomicidad**: Todo o nada. Si falla una operación, se deshacen todas.
- **Consistencia**: La base de datos pasa de un estado consistente a otro estado consistente.
- **Aislamiento**: Las transacciones concurrentes no interfieren entre sí.
- **Durabilidad**: Una vez confirmada (COMMIT), los cambios son permanentes.

4.2. Instrucciones de control de transacciones

- `START TRANSACTION` : Inicia una nueva transacción explícita.
- `COMMIT` o `COMMIT WORK` : Confirma los cambios de la transacción.
- `ROLLBACK` o `ROLLBACK WORK` : Deshace todos los cambios de la transacción.

5 AUTOCOMMIT

MariaDB tiene una variable de sesión llamada `AUTOCOMMIT` que controla si cada instrucción SQL se ejecuta automáticamente en su propia transacción.

5.1. Comportamiento por defecto (AUTOCOMMIT=1)

Por defecto, `AUTOCOMMIT` está activado (`AUTOCOMMIT=1`), lo que significa que:

- Cada instrucción SQL individual se ejecuta en su propia transacción.
- Después de cada instrucción exitosa, se hace un COMMIT automático.
- No es necesario usar `START TRANSACTION` , `COMMIT` o `ROLLBACK` explícitamente.

Verificar el estado de AUTOCOMMIT:

```
-- Ver el valor actual de AUTOCOMMIT
SELECT @@AUTOCOMMIT;
-- Resultado: 1 (activado) o 0 (desactivado)
```

5.2. Ejemplo con AUTOCOMMIT activado

Ejecuta el siguiente código que inserta tres notas, siendo la tercera errónea (valor negativo que viola la restricción CHECK):

```
-- Verificar que AUTOCOMMIT está activado
SET AUTOCOMMIT=1;

-- Insertar tres notas (la tercera fallará)
INSERT INTO grades (student_id, group_id, grade_value, exam_call,
with_honors)
VALUES (6, 1, 4.5, 'Primera', 0);

INSERT INTO grades (student_id, group_id, grade_value, exam_call,
with_honors)
VALUES (7, 1, 7.5, 'Primera', 0);

-- Esta instrucción fallará (valor negativo)
INSERT INTO grades (student_id, group_id, grade_value, exam_call,
with_honors)
VALUES (8, 1, -7.5, 'Primera', 0);
```

Resultado:

- Las dos primeras notas **SÍ se insertan** (cada una en su propia transacción con COMMIT automático).
- La tercera falla y no se inserta (su transacción hace ROLLBACK automático).
- Cada instrucción es independiente de las demás.

Verificar:

```
-- Ver las notas insertadas
SELECT * FROM grades
WHERE student_id IN (6, 7, 8) AND exam_call = 'Primera'
ORDER BY student_id;
-- Verás las dos primeras notas insertadas
```

5.3. Desactivar AUTOCOMMIT para transacciones explícitas

Cuando necesitamos agrupar varias instrucciones en una única transacción atómica, debemos:

1. Desactivar AUTOCOMMIT con `SET AUTOCOMMIT=0`
2. Usar `START TRANSACTION` para iniciar la transacción
3. Ejecutar las operaciones
4. Finalizar con `COMMIT` (confirmar) o `ROLLBACK` (cancelar)

Ejemplo:

```
-- Primero eliminar las notas del ejemplo anterior
DELETE FROM grades WHERE student_id IN (6, 7, 8) AND exam_call = 'Primera';
```

```
-- Desactivar AUTOCOMMIT
SET AUTOCOMMIT=0;

-- Iniciar transacción explícita
START TRANSACTION;

-- Insertar tres notas (la tercera fallará)
INSERT INTO grades (student_id, group_id, grade_value, exam_call,
with_honors)
VALUES (6, 1, 4.5, 'Primera', 0);

INSERT INTO grades (student_id, group_id, grade_value, exam_call,
with_honors)
VALUES (7, 1, 7.5, 'Primera', 0);

-- Esta instrucción fallará (valor negativo)
INSERT INTO grades (student_id, group_id, grade_value, exam_call,
with_honors)
VALUES (8, 1, -7.5, 'Primera', 0);

COMMIT;
```

Problema: El código anterior no funciona como esperamos porque:

- La tercera instrucción genera un error.
- El error detiene la ejecución del script.
- **Nunca se ejecuta** el `COMMIT` ni un posible `ROLLBACK`.
- La transacción queda activa con las dos primeras inserciones pendientes de confirmar.

Verificar:

```
-- Consultar dentro de la misma sesión (verás las dos notas)
SELECT * FROM grades
WHERE student_id IN (6, 7, 8) AND exam_call = 'Primera'
ORDER BY student_id;

-- Hacer ROLLBACK manual para deshacer
ROLLBACK;

-- Consultar de nuevo (ya no habrá notas)
SELECT * FROM grades
WHERE student_id IN (6, 7, 8) AND exam_call = 'Primera'
ORDER BY student_id;

-- Restaurar AUTOCOMMIT
SET AUTOCOMMIT=1;
```

5.4. Limitaciones del control de transacciones en scripts

¿Por qué no funciona el ROLLBACK en el script?

En un script SQL simple sin manejo de excepciones:

- Si ocurre un error, la ejecución del script se detiene inmediatamente.
- El código posterior al error (incluyendo ROLLBACK o COMMIT) **nunca se ejecuta**.
- La transacción queda activa y no se deshacen los cambios anteriores al error.

Solución: Para lograr un verdadero control transaccional con manejo de errores, debemos usar **procedimientos almacenados con manejadores de excepciones**.

6

Control de transacciones en procedimientos almacenados

Los procedimientos almacenados permiten implementar transacciones robustas con manejo de excepciones mediante `DECLARE EXIT HANDLER`.

6.1. Ejemplo: Procedimiento no transaccional

Crearemos un procedimiento que inserta un nuevo grado y una asignatura asociada. Sin control transaccional, si falla la inserción de la asignatura, el grado quedará insertado (inconsistencia).

```
DELIMITER //
CREATE OR REPLACE PROCEDURE p_insert_degree_subject(
    IN p_degree_name VARCHAR(80),
    IN p_degree_years TINYINT,
    IN p_subject_name VARCHAR(120),
    IN p_subject_acronym VARCHAR(12),
    IN p_subject_credits TINYINT,
    IN p_subject_course TINYINT,
    IN p_subject_type VARCHAR(30)
)
BEGIN
    DECLARE v_new_degree_id INT;

    -- Insertar el grado
    INSERT INTO degrees (degree_name, duration_years)
```

```
    VALUES (p_degree_name, p_degree_years);

    -- Obtener el ID del grado recién insertado
    SET v_new_degree_id = LAST_INSERT_ID();

    -- Insertar la asignatura asociada
    INSERT INTO subjects (degree_id, subject_name, acronym, credits,
course, subject_type)
    VALUES (v_new_degree_id, p_subject_name, p_subject_acronym,
            p_subject_credits, p_subject_course, p_subject_type);
END //
DELIMITER ;
```

Probar con datos válidos:

```
-- Inserción exitosa
CALL p_insert_degree_subject(
    'Grado en Inteligencia Artificial',
    4,
    'Aprendizaje Automático',
    'AA',
    6,
    3,
    'Obligatoria'
);

-- Verificar
SELECT * FROM degrees WHERE degree_name = 'Grado en Inteligencia
Artificial';
SELECT * FROM subjects WHERE acronym = 'AA';
```

Probar con datos erróneos:

```
-- Intentar insertar con nombre de grado duplicado (violará restricción
UNIQUE)
CALL p_insert_degree_subject(
    'Grado en Inteligencia Artificial',  -- Nombre duplicado
    4,
    'Redes Neuronales',
    'RN',
    6,
    3,
    'Obligatoria'
);

-- Verificar el problema
SELECT * FROM subjects WHERE acronym = 'RN';
-- ¡La asignatura SÍ se insertó aunque el grado falló! (PROBLEMA)
```

```
**Problema identificado:**
- La primera instrucción (INSERT del grado) falla porque el nombre ya
existe.
- El procedimiento termina con error.
- Sin embargo, si el error ocurriera en la segunda instrucción (INSERT de
la asignatura), el grado quedaría insertado.
- **Inconsistencia potencial**: operaciones interdependientes no se
ejecutan atómicamente.

**Limpiar:**

```sql
-- Eliminar datos de prueba
DELETE FROM subjects WHERE degree_id IN (
    SELECT degree_id FROM degrees
    WHERE degree_name IN ('Grado en Inteligencia Artificial', 'Grado en
Ciberseguridad')
);
DELETE FROM degrees WHERE degree_name IN ('Grado en Inteligencia
Artificial', 'Grado en Ciberseguridad');
```

6.2. Ejemplo: Procedimiento transaccional

Ahora implementaremos una versión transaccional que garantiza atomicidad: o se insertan ambos (grado y asignatura) o ninguno.

```
DELIMITER //
CREATE OR REPLACE PROCEDURE p_insert_degree_subject_transactional(
    IN p_degree_name VARCHAR(80),
    IN p_degree_years TINYINT,
    IN p_subject_name VARCHAR(120),
    IN p_subject_acronym VARCHAR(12),
    IN p_subject_credits TINYINT,
    IN p_subject_course TINYINT,
    IN p_subject_type VARCHAR(30)
)
BEGIN
    -- Iniciar la transacción
    START TRANSACTION;

    -- Bloque con manejo de excepciones
    tblock: BEGIN
        DECLARE v_new_degree_id INT;

        -- Declarar manejador de excepciones
        DECLARE EXIT HANDLER FOR SQLEXCEPTION, SQLWARNING
        BEGIN
```
```

```
            -- Si ocurre un error, deshacer todos los cambios
            ROLLBACK;
            -- Relanzar la excepción para que el llamador la vea
            RESIGNAL;
        END;

        -- Insertar el grado
        INSERT INTO degrees (degree_name, duration_years)
        VALUES (p_degree_name, p_degree_years);

        -- Obtener el ID del grado recién insertado
        SET v_new_degree_id = LAST_INSERT_ID();

        -- Insertar la asignatura asociada
        INSERT INTO subjects (degree_id, subject_name, acronym, credits,
course, subject_type)
        VALUES (v_new_degree_id, p_subject_name, p_subject_acronym,
                p_subject_credits, p_subject_course, p_subject_type);

        -- Si llegamos aquí, todo fue exitoso: confirmar cambios
        COMMIT;
    END tblock;
END //
DELIMITER ;
```

Observe lo siguiente:

- `START TRANSACTION`: Inicia la transacción al principio del procedimiento.
- `tblock: BEGIN ... END`: Bloque etiquetado necesario para definir el manejador de excepciones.
- `DECLARE EXIT HANDLER FOR SQLEXCEPTION, SQLWARNING`: Define qué hacer si ocurre un error o advertencia.
- `ROLLBACK`: Deshace todos los cambios de la transacción si hay error.
- `RESIGNAL`: Relanza la excepción para que el código que llamó al procedimiento la vea.
- `COMMIT`: Solo se ejecuta si no hubo errores, confirmando todos los cambios.

Probar con datos válidos:

```
-- Inserción exitosa
CALL p_insert_degree_subject_transactional(
    'Grado en Inteligencia Artificial',
    4,
    'Aprendizaje Automático',
    'AA',
    6,
    3,
    'Obligatoria'
);

-- Verificar que ambos se insertaron
```

```
SELECT * FROM degrees WHERE degree_name = 'Grado en Inteligencia
Artificial';
SELECT * FROM subjects WHERE acronym = 'AA';
```

Probar con datos erróneos:

```
-- Intentar insertar con nombre de grado duplicado
CALL p_insert_degree_subject_transactional(
    'Grado en Inteligencia Artificial',  -- Nombre duplicado (violará
UNIQUE)
    4,
    'Redes Neuronales',
    'RN',
    6,
    3,
    'Obligatoria'
);
-- Obtendrás un error y se hace ROLLBACK automático

-- Verificar que NO se insertó nada nuevo
SELECT * FROM subjects WHERE acronym = 'RN';
-- ¡La asignatura NO se insertó! (CORRECTO - atomicidad garantizada)
```

Resultado:

- Si ambas operaciones son exitosas: se hace COMMIT y ambos datos quedan persistentes.
- Si cualquier operación falla: se hace ROLLBACK automático y NO se persiste nada.
- **Atomicidad garantizada**: todo o nada.

6.3. Diferencias entre versiones

Aspecto	Sin transacción	Con transacción
Control de errores	No hay	Manejador de excepciones
Atomicidad	No garantizada	Garantizada
Si falla asignatura	Grado queda insertado	Nada se inserta
ROLLBACK automático	No	Sí
Consistencia	Puede violarse	Se mantiene

7 Transacciones concurrentes

Las transacciones proporcionan **aislamiento**: los cambios no confirmados (sin COMMIT) no son visibles para otras sesiones/transacciones.

7.1. Demostración con HeidiSQL

Abriremos dos ventanas de consulta en HeidiSQL para simular dos sesiones concurrentes.

Ventana 1 (Sesión A) - Iniciar transacción sin confirmar:

```
-- Desactivar AUTOCOMMIT para control manual
SET AUTOCOMMIT=0;

-- Iniciar transacción
START TRANSACTION;

-- Insertar dos notas
INSERT INTO grades (student_id, group_id, grade_value, exam_call,
with_honors)
VALUES (10, 1, 8.5, 'Primera', 0);

INSERT INTO grades (student_id, group_id, grade_value, exam_call,
with_honors)
VALUES (11, 1, 9.2, 'Primera', 1);

-- Consultar dentro de la transacción
SELECT g.grade_id, p.first_name, p.last_name, g.grade_value, g.with_honors
```

```
FROM grades g
JOIN students s ON s.student_id = g.student_id
JOIN people p ON p.person_id = s.student_id
WHERE g.student_id IN (10, 11);
```

Resultado en Sesión A:

- Verás las dos notas recién insertadas (están en tu transacción activa).

Ventana 2 (Sesión B) - Consultar desde otra sesión:

```
-- Consultar las mismas notas desde otra sesión
SELECT g.grade_id, p.first_name, p.last_name, g.grade_value, g.with_honors
FROM grades g
JOIN students s ON s.student_id = g.student_id
JOIN people p ON p.person_id = s.student_id
WHERE g.student_id IN (10, 11);
```

Resultado en Sesión B:

- **NO verás las notas nuevas** porque no se ha hecho COMMIT en la Sesión A.
- El aislamiento de transacciones impide que otras sesiones vean cambios no confirmados.

Volver a Ventana 1 (Sesión A) - Confirmar cambios:

```
-- Confirmar la transacción
COMMIT;

-- Restaurar AUTOCOMMIT
SET AUTOCOMMIT=1;
```

Volver a Ventana 2 (Sesión B) - Consultar de nuevo:

```
-- Consultar de nuevo
SELECT g.grade_id, p.first_name, p.last_name, g.grade_value, g.with_honors
FROM grades g
JOIN students s ON s.student_id = g.student_id
JOIN people p ON p.person_id = s.student_id
WHERE g.student_id IN (10, 11);
```

Resultado en Sesión B:

- **Ahora SÍ verás las notas** porque se hizo COMMIT en la Sesión A.
- Los cambios confirmados son visibles para todas las sesiones.

7.2. Niveles de aislamiento

MariaDB soporta diferentes niveles de aislamiento que determinan qué tan "aisladas" están las transacciones concurrentes:

- `READ UNCOMMITTED`: Permite lecturas sucias (leer datos no confirmados).
- `READ COMMITTED`: Solo lee datos confirmados (evita lecturas sucias).
- `REPEATABLE READ`: Las lecturas son repetibles (no cambian durante la transacción). **Por defecto en MariaDB/MySQL**.
- `SERIALIZABLE`: Máximo aislamiento (transacciones completamente serializadas).

Ver el nivel de aislamiento actual:

```
SELECT @@transaction_isolation;
-- Resultado típico: REPEATABLE-READ
```

Cambiar el nivel de aislamiento (solo para la sesión actual):

```
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

8

Buenas prácticas con transacciones

1. **Usar transacciones explícitas para operaciones relacionadas:**
   - Si varias operaciones deben ser atómicas, agrúpalas en una transacción.
2. **Siempre manejar excepciones en procedimientos transaccionales:**
   - Usar `DECLARE EXIT HANDLER FOR SQLEXCEPTION` con `ROLLBACK`.
3. **Mantener transacciones cortas:**
   - Las transacciones largas pueden bloquear recursos y afectar el rendimiento.
4. **Confirmar o revertir explícitamente:**
   - No dejar transacciones abiertas indefinidamente.
5. **Restaurar AUTOCOMMIT después de pruebas:**
   - Si desactivas AUTOCOMMIT para pruebas, recuerda reactivarlo: `SET AUTOCOMMIT=1`.
6. **Usar RESIGNAL en manejadores de excepciones:**
   - Permite que el código llamador sea consciente del error.
7. **Documentar procedimientos transaccionales:**
   - Indicar claramente qué procedimientos manejan transacciones.

9 Ejercicios propuestos

1. **Ejercicio 1**: Crea un procedimiento transaccional `p_transfer_student` que mueva un estudiante de un grupo a otro (borrar de `group_enrollments` el grupo antiguo e insertar el nuevo), garantizando atomicidad.
2. **Ejercicio 2**: Implementa un procedimiento `p_update_grade_with_history` que actualice una nota y guarde el valor antiguo en una tabla de historial `grade_history`. Debe ser atómico.
3. **Ejercicio 3**: Crea un procedimiento transaccional `p_batch_insert_grades` que inserte múltiples notas (recibirlas como parámetros separados por comas) y garantice que o se insertan todas o ninguna.

> *i* **Info**
>
> Versión PDF disponible