Actividad Práctica Integrada: Sistema Bancario

Temas: Indexación, Transacciones y Recuperación

📋 Objetivo de la Actividad

Implementar y analizar un sistema bancario que integre tres conceptos fundamentales:

1. **Indexación y Asociación** (Tema 6): Optimizar consultas mediante índices
2. **Procesamiento Transaccional** (Tema 7): Garantizar propiedades ACID
3. **Sistemas de Recuperación** (Tema 8): Manejar fallos y recuperación

🛠️ Herramientas Necesarias

Software Requerido:

- **MySQL 8.0+**
- **Cliente SQL**: MySQL Workbench o línea de comandos (`mysql`)
- **Editor de texto**: Para crear scripts SQL

Instalación Rápida:

```
# Verificar instalación de MySQL
mysql --version

# Conectar a MySQL
mysql -u root -p

# O desde MySQL Workbench: crear nueva conexión
```

📖 Enunciado y Contexto de la Actividad

Escenario del Sistema

Imagina que trabajas como **Administrador de Bases de Datos (DBA)** en un banco que necesita implementar un **nuevo sistema de gestión de cuentas y transacciones**. El banco ha decidido migrar de su sistema legacy a una solución moderna basada en MySQL.

Situación actual:

- El banco maneja aproximadamente 10,000 clientes activos
- Se procesan alrededor de 5,000 transacciones diarias
- El sistema actual tiene problemas de rendimiento en consultas frecuentes
- Se han reportado inconsistencias en transferencias cuando hay fallos del sistema

- No existe un sistema robusto de recuperación ante desastres

Requisitos del nuevo sistema:

1. **Rendimiento:** Las consultas de saldo y búsqueda de cuentas deben ser instantáneas
2. **Confiabilidad:** Todas las transacciones deben garantizar integridad de datos (ACID)
3. **Recuperación:** El sistema debe poder recuperarse automáticamente ante fallos
4. **Escalabilidad:** Debe soportar el crecimiento futuro del banco

Tu misión: Diseñar e implementar la base de datos del nuevo sistema aplicando los conceptos de **indexación**, **transacciones** y **recuperación** de manera integrada.

🎯 Metodología: Implementación de un Sistema Nuevo

Antes de comenzar con la implementación técnica, es fundamental seguir una metodología estructurada. A continuación se presenta el **paso a paso conceptual** para implementar un sistema de base de datos desde cero:

Fase 1: Análisis y Planificación

1.1 Identificación de Requisitos Funcionales

Objetivo: Entender qué debe hacer el sistema

Actividades:

- Reuniones con usuarios finales (cajeros, gerentes, clientes)
- Análisis de procesos de negocio actuales
- Identificación de operaciones críticas:
  - Consulta de saldos
  - Transferencias entre cuentas
  - Depósitos y retiros
  - Consulta de historial de transacciones
  - Búsqueda de clientes por DNI

Resultado: Documento de requisitos funcionales

1.2 Identificación de Requisitos No Funcionales

Objetivo: Definir características de calidad del sistema

Actividades:

- **Rendimiento:** Tiempo de respuesta < 100ms para consultas simples
- **Disponibilidad:** Sistema debe estar operativo 24/7
- **Confiabilidad:** Tolerancia a fallos con recuperación automática
- **Seguridad:** Control de acceso y auditoría de operaciones
- **Escalabilidad:** Capacidad de crecer sin cambios arquitectónicos

Resultado: Especificación de requisitos no funcionales

Fase 2: Diseño Conceptual

2.1 Identificación de Entidades

Objetivo: Identificar los objetos principales del dominio

Proceso:

1. **Análisis de sustantivos** en los requisitos funcionales
2. **Identificación de entidades principales:**
    - **Cliente:** Persona que tiene cuentas en el banco
    - **Cuenta:** Producto bancario asociado a un cliente
    - **Transacción:** Operación financiera que modifica saldos
    - **Log de Transacciones:** Registro de auditoría para recuperación
3. **Identificación de entidades relacionadas:**
    - Tipos de cuenta (Ahorro, Corriente)
    - Tipos de transacción (Depósito, Retiro, Transferencia)
    - Estados de transacción (Pendiente, Confirmada, Abortada)

Resultado: Diagrama de entidades principales

2.2 Identificación de Atributos

Objetivo: Definir las propiedades de cada entidad

Proceso por entidad:

Cliente:

- Identificadores: `cliente_id` (PK), `dni` (único)
- Datos personales: `nombre`, `apellido`, `email`, `telefono`
- Metadatos: `fecha_registro`

Cuenta:

- Identificadores: `cuenta_id` (PK), `numero_cuenta` (único)
- Relaciones: `cliente_id` (FK)
- Datos financieros: `tipo_cuenta`, `saldo`
- Metadatos: `fecha_apertura`, `activa`

Transacción:

- Identificadores: `transaccion_id` (PK)
- Relaciones: `cuenta_origen_id` (FK), `cuenta_destino_id` (FK)
- Datos financieros: `tipo_transaccion`, `monto`
- Metadatos: `fecha_transaccion`, `estado`, `descripcion`

Log de Transacciones:

- Identificadores: `log_id` (PK)
- Relaciones: `transaccion_id` (FK)
- Datos de auditoría: `operacion`, `tabla_afectada`, `registro_id`
- Datos de recuperación: `valor_anterior`, `valor_nuevo`, `timestamp_log`

Resultado: Especificación completa de atributos

2.3 Identificación de Relaciones

Objetivo: Definir cómo se relacionan las entidades

Relaciones identificadas:

- **Cliente** → **Cuenta:** 1 a N (un cliente puede tener múltiples cuentas)
- **Cuenta** → **Transacción (origen):** 1 a N (una cuenta puede tener múltiples transacciones como origen)
- **Cuenta** → **Transacción (destino):** 1 a N (una cuenta puede tener múltiples transacciones como destino)
- **Transacción** → **Log:** 1 a N (una transacción genera múltiples registros de log)

Resultado: Diagrama de relaciones (ER)

Fase 3: Diseño Lógico

3.1 Normalización

Objetivo: Eliminar redundancias y garantizar integridad

Proceso:

- **Primera Forma Normal (1NF):** Eliminar grupos repetitivos
- **Segunda Forma Normal (2NF):** Eliminar dependencias parciales
- **Tercera Forma Normal (3NF):** Eliminar dependencias transitivas

Resultado: Esquema normalizado

3.2 Definición de Restricciones

Objetivo: Garantizar integridad de datos

Restricciones identificadas:

- **Claves primarias:** Identificadores únicos
- **Claves foráneas:** Integridad referencial
- **Unicidad:** `dni`, `numero_cuenta`
- **Dominios:** Valores permitidos (tipos de cuenta, estados)
- **Chequeos:** `saldo >= 0`, `monto > 0`

Resultado: Especificación de restricciones

Fase 4: Diseño Físico

4.1 Parametrización del SGBD

Objetivo: Configurar MySQL para el caso de uso específico

Parámetros críticos a configurar:

Motor de Almacenamiento:

```
-- Usar InnoDB para soporte transaccional
ENGINE=InnoDB
```

Configuración de Transacciones:

- `autocommit`: Control manual de transacciones
- `transaction_isolation`: Nivel de aislamiento (READ COMMITTED recomendado)
- `innodb_lock_wait_timeout`: Tiempo de espera para bloqueos

Configuración de Logging:

- `log_bin`: Habilitar binlog para recuperación
- `binlog_format`: Formato del binlog (ROW recomendado)
- `innodb_log_file_size`: Tamaño del log de InnoDB
- `innodb_flush_log_at_trx_commit`: Frecuencia de escritura del log

Configuración de Rendimiento:

- `innodb_buffer_pool_size`: Memoria para caché de datos
- `max_connections`: Número máximo de conexiones simultáneas
- `query_cache_size`: Caché de consultas (MySQL 5.7, removido en 8.0)

Resultado: Archivo de configuración `my.cnf` optimizado

4.2 Estrategia de Indexación

Objetivo: Optimizar consultas frecuentes

Proceso:

1. **Análisis de consultas frecuentes:**
    - Búsqueda por número de cuenta
    - Búsqueda de cliente por DNI
    - Consulta de transacciones por cuenta y fecha
    - Consulta de saldo por cliente
2. **Identificación de columnas candidatas:**
    - Columnas en cláusulas WHERE
    - Columnas en JOIN
    - Columnas en ORDER BY

    - Columnas en GROUP BY

3. **Diseño de índices:**

    - **Índices primarios:** Ya definidos (claves primarias)
    - **Índices únicos:** Para búsquedas exactas (`numero_cuenta`, `dni`)
    - **Índices compuestos:** Para consultas con múltiples condiciones
    - **Índices covering:** Para consultas que solo necesitan datos del índice

Resultado: Plan de indexación

4.3 Estrategia de Transacciones

Objetivo: Garantizar propiedades ACID

Diseño:

- **Atomicidad:** Usar `START TRANSACTION` / `COMMIT` / `ROLLBACK`
- **Consistencia:** Validaciones antes de confirmar (fondos suficientes, cuentas activas)
- **Aislamiento:** Bloqueos `FOR UPDATE` en lecturas críticas
- **Durabilidad:** Configuración de `innodb_flush_log_at_trx_commit = 1`

Resultado: Procedimientos almacenados transaccionales

4.4 Estrategia de Recuperación

Objetivo: Garantizar recuperación ante fallos

Componentes:

- **Log de transacciones personalizado:** Tabla `log_transacciones` para auditoría
- **Binlog de MySQL:** Para recuperación a nivel de sistema
- **Backups regulares:** Estrategia de backup y restore
- **Procedimientos de recuperación:** UNDO y REDO manuales

Resultado: Plan de recuperación y procedimientos

Fase 5: Implementación

5.1 Creación del Esquema

Actividades:

- Crear base de datos
- Crear tablas con restricciones
- Crear índices según el plan
- Insertar datos de prueba

5.2 Implementación de Lógica de Negocio

Actividades:

- Crear procedimientos almacenados para operaciones críticas
- Implementar validaciones de negocio
- Configurar logging de transacciones

5.3 Pruebas y Validación

Actividades:

- Pruebas de rendimiento (con y sin índices)
- Pruebas de transacciones (casos exitosos y fallidos)
- Pruebas de recuperación (simulación de fallos)

Fase 6: Optimización y Mantenimiento

6.1 Monitoreo

- Análisis de planes de ejecución
- Identificación de consultas lentas
- Monitoreo de uso de índices

6.2 Ajustes

- Crear índices adicionales si es necesario
- Eliminar índices no utilizados
- Ajustar parámetros del SGBD según rendimiento observado

Estructura de la Actividad

Parte 1: Configuración del Esquema y Análisis de Indexación (30 min)

Parte 2: Implementación de Transacciones ACID (30 min)

Parte 3: Simulación de Fallos y Recuperación (30 min)

Tiempo total estimado: 90 minutos

Parte 1: Configuración del Esquema y Análisis de Indexación

Objetivo

Crear el esquema de base de datos y analizar el impacto de los índices en el rendimiento de consultas.

Script de Creación del Esquema

```
-- ============================================
-- SISTEMA BANCARIO: ESQUEMA BASE
-- ============================================
```

```sql
-- Crear base de datos
CREATE DATABASE banco_practica;
USE banco_practica;

-- Tabla de Clientes
CREATE TABLE clientes (
    cliente_id INT AUTO_INCREMENT PRIMARY KEY,
    dni VARCHAR(20) NOT NULL,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    email VARCHAR(100),
    telefono VARCHAR(20),
    fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB;

-- Tabla de Cuentas
CREATE TABLE cuentas (
    cuenta_id INT AUTO_INCREMENT PRIMARY KEY,
    cliente_id INT NOT NULL,
    numero_cuenta VARCHAR(20) UNIQUE NOT NULL,
    tipo_cuenta VARCHAR(20) NOT NULL,  -- 'AHORRO', 'CORRIENTE'
    saldo DECIMAL(15,2) DEFAULT 0.00,
    fecha_apertura TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    activa BOOLEAN DEFAULT TRUE,
    FOREIGN KEY (cliente_id) REFERENCES clientes(cliente_id)
        ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB;

-- Tabla de Transacciones
CREATE TABLE transacciones (
    transaccion_id INT AUTO_INCREMENT PRIMARY KEY,
    cuenta_origen_id INT,
    cuenta_destino_id INT,
    tipo_transaccion VARCHAR(20) NOT NULL,  -- 'DEPOSITO', 'RETIRO',
'TRANSFERENCIA'
    monto DECIMAL(15,2) NOT NULL,
    descripcion TEXT,
    fecha_transaccion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    estado VARCHAR(20) DEFAULT 'PENDIENTE',  -- 'PENDIENTE', 'CONFIRMADA',
'ABORTADA'
    FOREIGN KEY (cuenta_origen_id) REFERENCES cuentas(cuenta_id)
        ON DELETE RESTRICT ON UPDATE CASCADE,
    FOREIGN KEY (cuenta_destino_id) REFERENCES cuentas(cuenta_id)
        ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB;

-- Tabla de Log de Transacciones (para recuperación)
CREATE TABLE log_transacciones (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    transaccion_id INT NOT NULL,
    operacion VARCHAR(20) NOT NULL,  -- 'BEGIN', 'UPDATE', 'COMMIT', 'ROLLBACK'
    tabla_afectada VARCHAR(50),
    registro_id INT,
    valor_anterior TEXT,
```

```
    valor_nuevo TEXT,
    timestamp_log TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (transaccion_id) REFERENCES transacciones(transaccion_id)
        ON DELETE RESTRICT ON UPDATE CASCADE
) ENGINE=InnoDB;

-- Insertar datos de prueba
INSERT INTO clientes (dni, nombre, apellido, email, telefono) VALUES
('12345678A', 'Juan', 'Pérez', 'juan.perez@email.com', '600123456'),
('87654321B', 'María', 'García', 'maria.garcia@email.com', '600654321'),
('11223344C', 'Carlos', 'López', 'carlos.lopez@email.com', '600789012'),
('55667788D', 'Ana', 'Martínez', 'ana.martinez@email.com', '600345678'),
('99887766E', 'Luis', 'Sánchez', 'luis.sanchez@email.com', '600901234');

INSERT INTO cuentas (cliente_id, numero_cuenta, tipo_cuenta, saldo) VALUES
(1, 'ACC-001', 'CORRIENTE', 5000.00),
(1, 'ACC-002', 'AHORRO', 10000.00),
(2, 'ACC-003', 'CORRIENTE', 3000.00),
(3, 'ACC-004', 'AHORRO', 7500.00),
(4, 'ACC-005', 'CORRIENTE', 2500.00),
(5, 'ACC-006', 'AHORRO', 15000.00);
```

Actividad 1.1: Análisis SIN Índices

```
-- ============================================
-- ACTIVIDAD 1.1: Consultas SIN índices
-- ============================================

-- Habilitar análisis de plan de ejecución
-- MySQL: Usar EXPLAIN o EXPLAIN FORMAT=JSON para análisis detallado

-- Consulta 1: Búsqueda por número de cuenta
EXPLAIN FORMAT=JSON
SELECT * FROM cuentas WHERE numero_cuenta = 'ACC-001';

-- Consulta 2: JOIN con búsqueda por DNI
EXPLAIN FORMAT=JSON
SELECT c.*, cl.nombre, cl.apellido
FROM cuentas c
JOIN clientes cl ON c.cliente_id = cl.cliente_id
WHERE cl.dni = '12345678A';

-- Consulta 3: Búsqueda por cuenta y rango de fechas
EXPLAIN FORMAT=JSON
SELECT * FROM transacciones
WHERE cuenta_origen_id = 1
  AND fecha_transaccion BETWEEN '2024-01-01' AND '2024-12-31'
ORDER BY fecha_transaccion DESC;

-- Versión simple de EXPLAIN (más legible)
EXPLAIN
```

```
SELECT * FROM cuentas WHERE numero_cuenta = 'ACC-001';

-- Analizar resultados:
-- - type: ALL = Full Table Scan (sin índice)
-- - rows: Número de filas examinadas
-- - Extra: "Using where" indica filtrado sin índice
```

Preguntas para reflexión:

- ¿Qué tipo de escaneo realiza el SGBD? (type: ALL = Full Table Scan)
- ¿Cuántas filas se examinan? (columna `rows`)
- ¿Hay algún índice siendo utilizado? (columna `key` será NULL)

Actividad 1.2: Creación de Índices

```
-- ============================================
-- ACTIVIDAD 1.2: Crear índices estratégicos
-- ============================================

-- Índice único en número de cuenta (búsquedas frecuentes)
CREATE UNIQUE INDEX idx_cuentas_numero ON cuentas(numero_cuenta);

-- Índice en DNI de clientes (búsquedas por identificación)
CREATE INDEX idx_clientes_dni ON clientes(dni);

-- Índice compuesto en transacciones (búsquedas por cuenta y fecha)
-- MySQL 8.0+ soporta índices con ordenamiento DESC
CREATE INDEX idx_transacciones_cuenta_fecha
ON transacciones(cuenta_origen_id, fecha_transaccion DESC);

-- Índice covering para consultas de saldo (incluye columnas frecuentes)
CREATE INDEX idx_cuentas_cliente_tipo_saldo
ON cuentas(cliente_id, tipo_cuenta, saldo);

-- Índice en estado de transacciones (filtros frecuentes)
CREATE INDEX idx_transacciones_estado ON transacciones(estado);

-- Ver índices creados
SHOW INDEXES FROM cuentas;
SHOW INDEXES FROM clientes;
SHOW INDEXES FROM transacciones;

-- Ver información detallada de índices
SELECT
    TABLE_NAME,
    INDEX_NAME,
    COLUMN_NAME,
    SEQ_IN_INDEX,
    COLLATION,
    CARDINALITY
FROM information_schema.STATISTICS
```

```
WHERE TABLE_SCHEMA = 'banco_practica'
ORDER BY TABLE_NAME, INDEX_NAME, SEQ_IN_INDEX;
```

Actividad 1.3: Comparación de Rendimiento

```
-- ============================================
-- ACTIVIDAD 1.3: Comparar rendimiento CON índices
-- ============================================

-- Ejecutar las mismas consultas y comparar
EXPLAIN FORMAT=JSON
SELECT * FROM cuentas WHERE numero_cuenta = 'ACC-001';

EXPLAIN FORMAT=JSON
SELECT c.*, cl.nombre, cl.apellido
FROM cuentas c
JOIN clientes cl ON c.cliente_id = cl.cliente_id
WHERE cl.dni = '12345678A';

EXPLAIN FORMAT=JSON
SELECT * FROM transacciones
WHERE cuenta_origen_id = 1
  AND fecha_transaccion BETWEEN '2024-01-01' AND '2024-12-31'
ORDER BY fecha_transaccion DESC;

-- Versión simple para comparación rápida
EXPLAIN
SELECT * FROM cuentas WHERE numero_cuenta = 'ACC-001';

-- Analizar mejoras:
-- - type: ref o const (mejor que ALL)
-- - key: nombre del índice utilizado
-- - rows: debería ser 1 o muy pocas filas
-- - Extra: "Using index" indica index-only scan
```

Análisis de resultados:

- Comparar `type`: ALL vs ref/const
- Verificar `key`: NULL vs nombre del índice
- Comparar `rows`: muchas vs pocas filas
- Calcular mejora de rendimiento (porcentaje)

💰 Parte 2: Implementación de Transacciones ACID

Objetivo

Implementar operaciones bancarias que garanticen las propiedades ACID.

Actividad 2.1: Transferencia Bancaria con Transacciones

```
-- ============================================
-- ACTIVIDAD 2.1: Transferencia Bancaria
-- ============================================

-- Cambiar delimitador para procedimientos almacenados
DELIMITER //

-- Procedimiento para registrar en log
CREATE PROCEDURE registrar_log(
    IN p_transaccion_id INT,
    IN p_operacion VARCHAR(20),
    IN p_tabla_afectada VARCHAR(50),
    IN p_registro_id INT,
    IN p_valor_anterior TEXT,
    IN p_valor_nuevo TEXT
)
BEGIN
    INSERT INTO log_transacciones (
        transaccion_id, operacion, tabla_afectada,
        registro_id, valor_anterior, valor_nuevo
    ) VALUES (
        p_transaccion_id, p_operacion, p_tabla_afectada,
        p_registro_id, p_valor_anterior, p_valor_nuevo
    );
END //

-- Procedimiento de transferencia bancaria
CREATE PROCEDURE transferir_fondos(
    IN p_cuenta_origen INT,
    IN p_cuenta_destino INT,
    IN p_monto DECIMAL(15,2),
    OUT p_transaccion_id INT
)
BEGIN
    DECLARE v_saldo_origen DECIMAL(15,2);
    DECLARE v_saldo_destino DECIMAL(15,2);
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        -- Rollback automático en caso de error
        ROLLBACK;

        -- Marcar transacción como abortada si existe
        IF p_transaccion_id IS NOT NULL THEN
            UPDATE transacciones
            SET estado = 'ABORTADA'
            WHERE transaccion_id = p_transaccion_id;

            CALL registrar_log(p_transaccion_id, 'ROLLBACK', NULL, NULL, NULL,
NULL);
        END IF;

        -- Re-lanzar el error
```

```
        RESIGNAL;
    END;

    -- Iniciar transacción
    START TRANSACTION;

    -- 1. Crear registro de transacción
    INSERT INTO transacciones (
        cuenta_origen_id, cuenta_destino_id,
        tipo_transaccion, monto, estado
    ) VALUES (
        p_cuenta_origen, p_cuenta_destino,
        'TRANSFERENCIA', p_monto, 'PENDIENTE'
    );

    SET p_transaccion_id = LAST_INSERT_ID();

    -- Registrar BEGIN en log
    CALL registrar_log(p_transaccion_id, 'BEGIN', NULL, NULL, NULL, NULL);

    -- 2. Verificar fondos (con bloqueo FOR UPDATE)
    SELECT saldo INTO v_saldo_origen
    FROM cuentas
    WHERE cuenta_id = p_cuenta_origen
    FOR UPDATE;  -- Bloqueo exclusivo

    IF v_saldo_origen < p_monto THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = CONCAT('Fondos insuficientes. Saldo actual: ',
v_saldo_origen);
    END IF;

    -- Registrar lectura en log
    CALL registrar_log(
        p_transaccion_id, 'READ', 'cuentas',
        p_cuenta_origen,
        CAST(v_saldo_origen AS CHAR), NULL
    );

    -- 3. Obtener saldo destino
    SELECT saldo INTO v_saldo_destino
    FROM cuentas
    WHERE cuenta_id = p_cuenta_destino
    FOR UPDATE;

    CALL registrar_log(
        p_transaccion_id, 'READ', 'cuentas',
        p_cuenta_destino,
        CAST(v_saldo_destino AS CHAR), NULL
    );

    -- 4. Debitar cuenta origen
    UPDATE cuentas
    SET saldo = saldo - p_monto
```

```
    WHERE cuenta_id = p_cuenta_origen;

    CALL registrar_log(
        p_transaccion_id, 'UPDATE', 'cuentas',
        p_cuenta_origen,
        CAST(v_saldo_origen AS CHAR),
        CAST((v_saldo_origen - p_monto) AS CHAR)
    );

    -- 5. Acreditar cuenta destino
    UPDATE cuentas
    SET saldo = saldo + p_monto
    WHERE cuenta_id = p_cuenta_destino;

    CALL registrar_log(
        p_transaccion_id, 'UPDATE', 'cuentas',
        p_cuenta_destino,
        CAST(v_saldo_destino AS CHAR),
        CAST((v_saldo_destino + p_monto) AS CHAR)
    );

    -- 6. Confirmar transacción
    UPDATE transacciones
    SET estado = 'CONFIRMADA'
    WHERE transaccion_id = p_transaccion_id;

    CALL registrar_log(p_transaccion_id, 'COMMIT', NULL, NULL, NULL, NULL);

    -- Confirmar transacción
    COMMIT;
END //

-- Restaurar delimitador
DELIMITER ;

-- Ejecutar transferencia exitosa
SET @trans_id = 0;
CALL transferir_fondos(1, 2, 500.00, @trans_id);
SELECT @trans_id AS transaccion_id;

-- Verificar saldos
SELECT cuenta_id, numero_cuenta, saldo FROM cuentas WHERE cuenta_id IN (1, 2);

-- Ver log de transacciones
SELECT * FROM log_transacciones
WHERE transaccion_id = @trans_id
ORDER BY timestamp_log;
```

Actividad 2.2: Simular Transacción que Falla

```
-- ============================================
-- ACTIVIDAD 2.2: Transacción que falla (fondos insuficientes)
-- ============================================

-- Intentar transferir más de lo disponible
SET @trans_id = 0;
CALL transferir_fondos(3, 4, 5000.00, @trans_id);  -- Solo tiene 3000

-- Verificar que no se aplicaron cambios
SELECT cuenta_id, numero_cuenta, saldo FROM cuentas WHERE cuenta_id IN (3, 4);

-- Ver log de la transacción abortada
SELECT * FROM log_transacciones
WHERE transaccion_id = @trans_id
ORDER BY timestamp_log;

-- Ver estado de la transacción
SELECT * FROM transacciones WHERE transaccion_id = @trans_id;
```

Actividad 2.3: Verificar Propiedades ACID

```
-- ============================================
-- ACTIVIDAD 2.3: Verificar Atomicidad
-- ============================================

-- Simular fallo a mitad de transacción (manual)
START TRANSACTION;

    -- Operación 1: Debitar
    UPDATE cuentas SET saldo = saldo - 1000 WHERE cuenta_id = 1;

    -- Simular fallo (no ejecutar la siguiente línea)
    -- UPDATE cuentas SET saldo = saldo + 1000 WHERE cuenta_id = 2;

    -- Ver estado antes de commit
    SELECT cuenta_id, saldo FROM cuentas WHERE cuenta_id IN (1, 2);

-- Si hacemos ROLLBACK, todo se revierte
ROLLBACK;

-- Verificar que se revirtió todo
SELECT cuenta_id, saldo FROM cuentas WHERE cuenta_id IN (1, 2);

-- Verificar aislamiento: abrir dos conexiones y probar bloqueos
-- Conexión 1:
START TRANSACTION;
SELECT saldo FROM cuentas WHERE cuenta_id = 1 FOR UPDATE;
-- No hacer COMMIT todavía

-- Conexión 2 (en otra ventana/terminal):
```

```
START TRANSACTION;
SELECT saldo FROM cuentas WHERE cuenta_id = 1 FOR UPDATE;
-- Esta consulta esperará hasta que la conexión 1 haga COMMIT o ROLLBACK
```

Parte 3: Simulación de Fallos y Recuperación

Objetivo

Simular fallos del sistema y demostrar cómo el log histórico permite la recuperación.

Actividad 3.1: Configurar Binlog y Logging

```
-- ============================================
-- ACTIVIDAD 3.1: Verificar configuración de Binlog
-- ============================================

-- Verificar que el binlog está habilitado
SHOW VARIABLES LIKE 'log_bin';
SHOW VARIABLES LIKE 'binlog_format';

-- Ver estado actual del binlog
SHOW MASTER STATUS;

-- Ver eventos del binlog recientes
SHOW BINLOG EVENTS LIMIT 20;

-- Verificar que nuestro log de transacciones está funcionando
SELECT COUNT(*) AS total_logs FROM log_transacciones;

-- Ver configuración de InnoDB (motor de almacenamiento transaccional)
SHOW VARIABLES LIKE 'innodb%log%';
SHOW VARIABLES LIKE 'innodb%checkpoint%';
```

Actividad 3.2: Simular Fallo y Recuperación Manual

```
-- ============================================
-- ACTIVIDAD 3.2: Simular fallo durante transacción
-- ============================================

-- Paso 1: Iniciar transacción
START TRANSACTION;

    -- Crear registro de transacción
    INSERT INTO transacciones (cuenta_origen_id, tipo_transaccion, monto, estado)
    VALUES (1, 'RETIRO', 200.00, 'PENDIENTE');

    SET @trans_id = LAST_INSERT_ID();
```

```
    -- Registrar BEGIN en log
    INSERT INTO log_transacciones (transaccion_id, operacion, tabla_afectada)
    VALUES (@trans_id, 'BEGIN', NULL);

    -- Operación 1: Leer saldo
    SELECT saldo INTO @saldo_anterior
    FROM cuentas WHERE cuenta_id = 1;

    INSERT INTO log_transacciones (transaccion_id, operacion, tabla_afectada,
registro_id, valor_anterior)
    VALUES (@trans_id, 'READ', 'cuentas', 1, CAST(@saldo_anterior AS CHAR));

    -- Operación 2: Modificar
    UPDATE cuentas SET saldo = saldo - 200 WHERE cuenta_id = 1;

    SELECT saldo INTO @saldo_nuevo FROM cuentas WHERE cuenta_id = 1;

    INSERT INTO log_transacciones (transaccion_id, operacion, tabla_afectada,
registro_id, valor_anterior, valor_nuevo)
    VALUES (@trans_id, 'UPDATE', 'cuentas', 1,
            CAST(@saldo_anterior AS CHAR),
            CAST(@saldo_nuevo AS CHAR));

    -- SIMULAR FALLO: No hacer COMMIT, cerrar conexión o reiniciar MySQL
    -- Simulamos dejando la transacción abierta

-- Si el sistema falla aquí, MySQL hará ROLLBACK automático al reiniciar
-- Pero simularemos la recuperación manual

-- Paso 2: Proceso de recuperación (después del fallo simulado)
-- Primero, hacer ROLLBACK para simular el fallo
ROLLBACK;

-- Identificar transacciones incompletas
SELECT DISTINCT transaccion_id
FROM log_transacciones
WHERE transaccion_id NOT IN (
    SELECT DISTINCT transaccion_id
    FROM log_transacciones
    WHERE operacion = 'COMMIT'
)
AND operacion = 'BEGIN';

-- Paso 3: UNDO - Deshacer cambios de transacciones incompletas
-- Procedimiento para UNDO
DELIMITER //

CREATE PROCEDURE undo_transaccion(IN p_transaccion_id INT)
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE v_registro_id INT;
    DECLARE v_valor_anterior TEXT;
    DECLARE v_tabla_afectada VARCHAR(50);
```

```
    DECLARE cur_undo CURSOR FOR
        SELECT registro_id, valor_anterior, tabla_afectada
        FROM log_transacciones
        WHERE transaccion_id = p_transaccion_id
        AND operacion = 'UPDATE'
        ORDER BY timestamp_log DESC;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur_undo;

    read_loop: LOOP
        FETCH cur_undo INTO v_registro_id, v_valor_anterior, v_tabla_afectada;

        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Restaurar valor anterior según la tabla
        IF v_tabla_afectada = 'cuentas' THEN
            UPDATE cuentas
            SET saldo = CAST(v_valor_anterior AS DECIMAL(15,2))
            WHERE cuenta_id = v_registro_id;

            SELECT CONCAT('UNDO: Restaurado cuenta ', v_registro_id, ' a valor ',
v_valor_anterior) AS mensaje;
        END IF;
    END LOOP;

    CLOSE cur_undo;

    -- Marcar transacción como abortada
    UPDATE transacciones
    SET estado = 'ABORTADA'
    WHERE transaccion_id = p_transaccion_id;

    INSERT INTO log_transacciones (transaccion_id, operacion)
    VALUES (p_transaccion_id, 'ROLLBACK');
END //

DELIMITER ;

-- Ejecutar UNDO para la transacción incompleta
CALL undo_transaccion(@trans_id);

-- Verificar recuperación
SELECT cuenta_id, saldo FROM cuentas WHERE cuenta_id = 1;
```

Actividad 3.3: Proceso de Redo (Reaplicar Transacciones Confirmadas)

```
-- ============================================
-- ACTIVIDAD 3.3: REDO - Reaplicar transacciones confirmadas
-- ============================================

-- Simular: Transacciones confirmadas que se perdieron en memoria
-- pero están en el log

-- Crear tabla temporal para simular pérdida de datos
CREATE TEMPORARY TABLE cuentas_backup AS SELECT * FROM cuentas;

-- "Perder" algunos cambios (simular)
UPDATE cuentas SET saldo = 0 WHERE cuenta_id = 1;

-- Procedimiento REDO
DELIMITER //

CREATE PROCEDURE redo_transacciones()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE v_transaccion_id INT;
    DECLARE v_registro_id INT;
    DECLARE v_valor_nuevo TEXT;
    DECLARE v_tabla_afectada VARCHAR(50);

    DECLARE cur_transacciones CURSOR FOR
        SELECT DISTINCT transaccion_id
        FROM log_transacciones
        WHERE operacion = 'COMMIT'
        AND transaccion_id IN (
            SELECT DISTINCT transaccion_id
            FROM log_transacciones
            WHERE operacion = 'UPDATE'
        );

    DECLARE cur_updates CURSOR FOR
        SELECT registro_id, valor_nuevo, tabla_afectada
        FROM log_transacciones
        WHERE transaccion_id = v_transaccion_id
        AND operacion = 'UPDATE'
        ORDER BY timestamp_log;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur_transacciones;

    trans_loop: LOOP
        FETCH cur_transacciones INTO v_transaccion_id;

        IF done THEN
            LEAVE trans_loop;
        END IF;

        SET done = FALSE;
```

```
        OPEN cur_updates;

        update_loop: LOOP
            FETCH cur_updates INTO v_registro_id, v_valor_nuevo, v_tabla_afectada;

            IF done THEN
                LEAVE update_loop;
            END IF;

            -- Reaplicar cambio según la tabla
            IF v_tabla_afectada = 'cuentas' THEN
                UPDATE cuentas
                SET saldo = CAST(v_valor_nuevo AS DECIMAL(15,2))
                WHERE cuenta_id = v_registro_id;

                SELECT CONCAT('REDO: Reaplicado cambio en cuenta ', v_registro_id,
' a valor ', v_valor_nuevo) AS mensaje;
            END IF;
        END LOOP;

        CLOSE cur_updates;
        SET done = FALSE;
    END LOOP;

    CLOSE cur_transacciones;
END //

DELIMITER ;

-- Ejecutar REDO
CALL redo_transacciones();

-- Verificar que se recuperaron los datos
SELECT cuenta_id, saldo FROM cuentas WHERE cuenta_id = 1;
SELECT cuenta_id, saldo FROM cuentas_backup WHERE cuenta_id = 1;
```

Actividad 3.4: Análisis del Log de Transacciones

```
-- ============================================
-- ACTIVIDAD 3.4: Análisis del log
-- ============================================

-- Ver todas las transacciones y su estado
SELECT
    t.transaccion_id,
    t.tipo_transaccion,
    t.monto,
    t.estado,
    COUNT(lt.log_id) AS num_operaciones,
    MIN(lt.timestamp_log) AS inicio,
```

```
    MAX(lt.timestamp_log) AS fin
FROM transacciones t
LEFT JOIN log_transacciones lt ON t.transaccion_id = lt.transaccion_id
GROUP BY t.transaccion_id, t.tipo_transaccion, t.monto, t.estado
ORDER BY t.transaccion_id;

-- Ver secuencia completa de una transacción específica
SELECT
    operacion,
    tabla_afectada,
    registro_id,
    valor_anterior,
    valor_nuevo,
    timestamp_log
FROM log_transacciones
WHERE transaccion_id = (SELECT MAX(transaccion_id) FROM transacciones)
ORDER BY timestamp_log;

-- Identificar transacciones incompletas (sin COMMIT ni ROLLBACK)
SELECT DISTINCT lt.transaccion_id
FROM log_transacciones lt
WHERE lt.operacion = 'BEGIN'
AND lt.transaccion_id NOT IN (
    SELECT DISTINCT transaccion_id
    FROM log_transacciones
    WHERE operacion IN ('COMMIT', 'ROLLBACK')
);

-- Análisis de rendimiento del log
SELECT
    operacion,
    COUNT(*) AS cantidad,
    MIN(timestamp_log) AS primera_operacion,
    MAX(timestamp_log) AS ultima_operacion
FROM log_transacciones
GROUP BY operacion
ORDER BY cantidad DESC;
```

📊 Parte 4: Análisis y Conclusiones

Preguntas de Reflexión

1. **Indexación:**
   - ¿Qué mejora de rendimiento observaste con los índices?
   - ¿Qué índices fueron más efectivos y por qué?
   - ¿Hay algún índice que no se esté utilizando? (verificar con `SHOW INDEXES`)
2. **Transacciones:**
   - ¿Cómo se garantizó la atomicidad en las transferencias?

- ¿Qué pasaría si no usáramos transacciones?
- ¿Cómo funcionan los bloqueos FOR UPDATE?

3. **Recuperación:**

- ¿Por qué es importante el orden de las operaciones en el log?
- ¿Cuál es la diferencia entre UNDO y REDO?
- ¿Qué información mínima necesita el log para recuperar el sistema?

Tareas Adicionales (Opcional)

1. **Optimización de Índices:**

- Crear índices covering para consultas específicas
- Analizar el uso de índices con `SHOW INDEXES` y estadísticas
- Usar `ANALYZE TABLE` para actualizar estadísticas

2. **Transacciones Avanzadas:**

- Implementar savepoints con `SAVEPOINT` y `ROLLBACK TO SAVEPOINT`
- Probar diferentes niveles de aislamiento: `SET TRANSACTION ISOLATION LEVEL`

3. **Recuperación Avanzada:**

- Configurar binlog para replicación
- Simular recuperación punto-en-tiempo usando binlog

☑ Checklist de Verificación

- [ ] Esquema de base de datos creado correctamente
- [ ] Índices creados y analizados su impacto
- [ ] Transacciones funcionando con propiedades ACID
- [ ] Log de transacciones registrando todas las operaciones
- [ ] Simulación de fallo y recuperación exitosa
- [ ] Análisis de rendimiento completado
- [ ] Conclusiones documentadas

Notas para el Instructor

Puntos Clave a Enfatizar:

1. **Indexación:**

- Los índices mejoran lecturas pero ralentizan escrituras
- No todos los índices son útiles; analizar antes de crear
- Los índices covering pueden eliminar acceso a tablas
- Usar `ANALYZE TABLE` para actualizar estadísticas

2. **Transacciones:**

    - START TRANSACTION/COMMIT/ROLLBACK son fundamentales
    - Los bloqueos FOR UPDATE previenen condiciones de carrera
    - El log debe escribirse ANTES de modificar datos (principio WAL)
    - InnoDB es el motor transaccional por defecto en MySQL

3. **Recuperación:**

    - El log es la fuente de verdad
    - UNDO para transacciones no confirmadas
    - REDO para transacciones confirmadas pero no escritas en disco
    - MySQL usa binlog para replicación y recuperación

Solución de Problemas Comunes:

- **Error de bloqueo:** Verificar que no hay transacciones abiertas con `SHOW PROCESSLIST`
- **Índices no utilizados:** Actualizar estadísticas con `ANALYZE TABLE nombre_tabla`
- **Log muy grande:** Configurar rotación de binlog o purgar logs antiguos
- **Procedimientos no se crean:** Verificar que el delimitador esté correctamente configurado

Comandos Útiles de MySQL:

```
-- Ver transacciones activas
SHOW PROCESSLIST;

-- Ver estado de InnoDB
SHOW ENGINE INNODB STATUS\G

-- Ver configuración de transacciones
SHOW VARIABLES LIKE 'transaction%';
SHOW VARIABLES LIKE 'isolation%';

-- Actualizar estadísticas de tablas
ANALYZE TABLE cuentas, clientes, transacciones;

-- Ver información de tablas
SHOW TABLE STATUS LIKE 'cuentas';
```

🎯 Resultado Esperado

Al finalizar esta actividad, los estudiantes deberían:

- Entender el impacto de los índices en el rendimiento
- Comprender cómo las transacciones garantizan ACID
- Saber cómo funciona la recuperación ante fallos
- Poder analizar y optimizar sistemas de bases de datos reales
- Dominar el uso de MySQL Workbench y línea de comandos

📌 Notas Adicionales para MySQL

Diferencias Clave con PostgreSQL:

1. **Auto-increment:** MySQL usa `AUTO_INCREMENT` en lugar de `SERIAL`
2. **Procedimientos:** Sintaxis diferente, requiere `DELIMITER`
3. **Manejo de errores:** Usa `DECLARE HANDLER` y `SIGNAL SQLSTATE`
4. **Binlog:** MySQL usa binlog en lugar de WAL (aunque InnoDB tiene su propio log)
5. **EXPLAIN:** MySQL no tiene `ANALYZE`, usa `EXPLAIN FORMAT=JSON` para más detalles

Configuración Recomendada para la Actividad:

```
-- Verificar configuración actual
SHOW VARIABLES LIKE 'autocommit';
SHOW VARIABLES LIKE 'transaction_isolation';

-- Configurar para mejor demostración (opcional)
SET autocommit = 0;  -- Desactivar autocommit para control manual
SET transaction_isolation = 'READ COMMITTED';  -- Nivel de aislamiento
```