Transacciones

Bases de Datos

Curso 2016-2017

Jesús Correas – jcorreas@ucm.es

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

.

Ejemplo introductorio

- En muchas ocasiones la realización de una operación en una BD requiere **modificar o insertar varios datos en múltiples tablas.**
- Para ello, es necesario ejecutar varias sentencias SQL.
- **Ejemplo: traspaso de dinero entre cuentas bancarias.**
- Supongamos que tenemos dos tablas con la siguiente estructura:

Tabla **Cuentas**

Cta	**Titular**	Saldo
37	27347234T	1500,00
44	85647456W	2300,00
...	...	...

Tabla **Movimientos**

Cta	**NumMto**	**Fecha**	**Importe**
37	1	25/03/2015	850,00
37	2	25/04/2015	-350,00
37	3	13/08/2016	1000,00
...	...	...	...
44	1	01/12/2016	500,00
44	2	28/12/2016	1800,00

- Para traspasar 400,00 euros entre las cuentas 37 y 44 es necesario:
  - Anotar el **movimiento negativo** en la cuenta de origen.
  - Anotar el **movimiento positivo** en la cuenta de destino.
  - Actualizar el **saldo** de las dos cuentas.

Ejemplo introductorio

- Las operaciones que se deben realizar son:

```
SELECT Saldo INTO vSaldo FROM Cuentas WHERE Cta = '37';
IF vSaldo > vImporte THEN
  INSERT INTO Movimientos VALUES ('37', 4, SYSDATE, -400.00);
  INSERT INTO Movimientos VALUES ('44', 3, SYSDATE,  400.00);
  UPDATE Cuentas SET Saldo = vSaldo - 400.00 WHERE Cta = '37';
  UPDATE Cuentas SET Saldo = Saldo + 400.00 WHERE Cta = '44';
END IF;
```

Tabla **Cuentas**

Cta	Titular	Saldo
37	27347234T	1100,00
44	85647456W	2700,00
...	...	...

Tabla **Movimientos**

Cta	NumMto	Fecha	Importe
37	1	25/03/2015	850,00
37	2	25/04/2015	-350,00
37	3	13/08/2016	1000,00
37	**4**	**09/01/2017**	**-400,00**
...	...	...	...
44	1	01/12/2016	500,00
44	2	28/12/2016	1800,00
44	**3**	**09/01/2017**	**400,00**

- El SGBD **debe garantizar** que el estado de la BD **es consistente: o se hacen todas las operaciones, o no se hace ninguna.**

Transacciones

- Hay muchas situaciones que podrían dejar la BD **inconsistente:**
  - Hay un corte de suministro eléctrico en el servidor de BD durante la ejecución de estas sentencias.
  - Hay un fallo en la máquina cliente.
  - Hay una interrupción en la red que une cliente y servidor.
  - Hay múltiples usuarios modificando la BD al mismo tiempo.
- Este último caso es especialmente importante, porque un SGBD puede tener **gran número de usuarios (o procesos) concurrentes.**
  - Un proceso se conecta al SGBD iniciando una **sesión,** que está activa mientras el proceso no la cierre o termine su ejecución.
- Una **transacción** es un conjunto de **una o varias instrucciones SQL cuya ejecución constituye una unidad lógica e indivisible.**
  - Una o más instrucciones DML que modifican la BD.
  - Una instrucción DDL.
- Las sentencias de una transacción se realizan todas o ninguna. No existen operaciones parciales.

Propiedades ACID

- El sistema de transacciones de cualquier BD relacional debe cumplir cuatro propiedades denominadas **ACID**:
  - **A de atomicidad:** En una transacción, todas las operaciones se terminan, o bien no se realiza ninguna.
  - **C de consistencia:** Una consulta debe ser consistente con el estado de la base de datos **en el instante de inicio** de la ejecución de la consulta.
  - **I de aislamiento:** Una transacción no completada es invisible al resto del mundo.
  - **D de durabilidad:** Cuando se completa una transacción, entonces es imposible que la base de datos la pierda.

Control de Transacciones: **COMMIT** y **ROLLBACK**

- Para ello existe un conjunto de instrucciones SQL que permiten implementar el control de las transacciones.
- **No existe una sentencia específica de inicio de transacción:**
  - Se inicia cuando se ejecuta una sentencia DML o DDL.
  - O cuando se ejecuta una sentencia **SET TRANSACTION.**
- Hay dos sentencias básicas para **finalizar una transacción:** **COMMIT** y **ROLLBACK**.
  - **COMMIT confirma** las modificaciones realizadas en la BD y los hace permanentes y visibles a las demás sesiones activas.
  - **ROLLBACK cancela todos los cambios** que se hayan realizado desde el inicio de la transacción.
- Además, cualquier sentencia DDL **termina implícitamente cualquier transacción** (confirmando las sentencias DML anteriores, como un `COMMIT`).
- Cuando termina una transacción, la siguiente instrucción SQL ejecutable inicia automáticamente la siguiente transacción.

Ejemplo

```
DECLARE
  vImporte NUMBER := 400;
  vExisteDestino NUMBER;
  vSaldo Cuentas.Saldo%TYPE;
BEGIN
  SELECT Saldo INTO vSaldo FROM Cuentas WHERE Cta = '37';
  IF vSaldo > vImporte THEN
    -- INICIO DE TRANSACCION. -- Cargo en la cta. de origen.
    INSERT INTO Movimientos VALUES ('37', 4, SYSDATE, -vImporte);
    UPDATE Cuentas SET Saldo = Saldo - vImporte WHERE Cta = '37';

    -- Si existe la cta. destino, abono en la cta. destino.
    SELECT COUNT(*) INTO vExisteDestino
    FROM Cuentas WHERE Cta = '44';
    IF vExisteDestino > 0 THEN
      INSERT INTO Movimientos VALUES ('44',3,SYSDATE,vImporte);
      UPDATE Cuentas SET Saldo = Saldo + vImporte WHERE Cta='44';
      COMMIT; -- FIN DE TRANSACCION: confirma los cambios.
    ELSE
      ROLLBACK; -- FIN DE TRANSACCION: deshace los cambios.
    END IF;
  END IF;
END;
```

Control de Transacciones: **SAVEPOINT**

- Además de las sentencias **COMMIT** y **ROLLBACK**, se pueden fijar **puntos intermedios:**
  - La instrucción ***SAVEPOINT identificador*** establece un punto en una transacción hasta el que se puede cancelar parcialmente.
  - Para hacerlo se utiliza la sentencia:

    ROLLBACK TO SAVEPOINT identificador.

- Sin embargo, los SAVEPOINT intermedios **no finalizan la transacción:** solo deshacen algunos de los cambios realizados.
- Se pueden utilizar **varios SAVEPOINT** en una misma transacción:
  - Cuando se retrocede a un SAVEPOINT, se eliminan todos los SAVEPOINT posteriores, pero no los anteriores.
  - Se puede **desplazar** un SAVEPOINT utilizando el mismo identificador en distintos puntos: se retrocede al último punto ejecutado con ese identificador.

Ejemplo

- **¿Cuál es el estado de la BD después de ejecutar las siguientes sentencias?**

```
CREATE TABLE empl (
  NIF VARCHAR2(9) PRIMARY KEY,
  NOMBRE VARCHAR2(20),
  SALARIO NUMBER(6,2)
);
INSERT INTO empl VALUES ('10A','Jorge Perez',3000.11);
ROLLBACK;
INSERT INTO empl VALUES ('30C','Javier Sala',2000.22);
INSERT INTO empl VALUES ('30C','Soledad Lopez',2000.33);
INSERT INTO empl VALUES ('40D','Sonia Moldes',1800.44);
INSERT INTO empl VALUES ('50E','Antonio Lopez',1800.44);
COMMIT;
INSERT INTO empl VALUES ('70C','Soledad Martin',2000.33);
```

- **¿Cuál es el estado de la BD visible desde otras sesiones?**

Otro ejemplo

- **¿Cuál es el estado de la BD después de ejecutar las siguientes sentencias?**

```
SET TRANSACTION NAME 'sal_update';
UPDATE empl SET salario = 7000 WHERE NIF= '30C';

SAVEPOINT after_salario;
UPDATE empl SET salario = 12000 WHERE NIF= '40D';

ROLLBACK TO SAVEPOINT after_salario;
UPDATE empl SET salario = 11000 WHERE NIF= '40D';
COMMIT;
```

Bloqueos

- La **propiedad de aislamiento** de ACID exige que dos transacciones no completadas **no puedan afectarse mutuamente.**
- Para proporcionar aislamiento, los SGBD utilizan técnicas de **bloqueo** de los elementos de la BD.
  - Un bloqueo **restringe el acceso a estos elementos para determinadas operaciones.**
- De forma simplificada, hay **dos tipos de bloqueo:**
  - **Bloqueos compartidos:** Otras sesiones pueden leer los datos bloqueados, pero **no pueden modificarlos.** Este bloqueo lo producen las sentencias de consulta (`SELECT`).
  - **Bloqueos exclusivos:** Otras sesiones no pueden acceder a los datos hasta que no se libera el bloqueo. Este bloqueo lo producen las sentencias DDL y de modificación (`INSERT`, `UPDATE`, `DELETE`).
  - Aunque son diferentes, consideramos **bloqueos exclusivos** los generados por cursores `SELECT...FOR UPDATE.`
- En un bloqueo exclusivo, otras sesiones pueden consultar el contenido de las tablas **sin los cambios de transacciones no confirmadas.**

Bloqueos

- Existe un **conflicto** cuando dos sesiones de acceso a Oracle **intentan modificar los mismos datos a la vez.**
- La base de datos debe **serializar** los accesos concurrentes, mediante estos mecanismos de **bloqueo de filas o de tablas completas.**
- Oracle intenta bloquear la menor cantidad de datos necesaria para obtener el máximo nivel de concurrencia.
  - Las sentencias **DDL** (`CREATE TABLE`, `ALTER TABLE`) bloquean **toda la tabla.**
  - Las sentencias **DML** de modificación de datos (`INSERT`, `UPDATE`, `DELETE`, `SELECT...FOR UPDATE`) bloquean **las filas afectadas.**
- El nivel de aislamiento se puede configurar con **`SET TRANSACTION.`**

Conflictos de bloqueo: Comportamiento ante el bloqueo

- Si una **sentencia DDL** no puede obtener el bloqueo sobre un objeto, entonces **termina inmediatamente con una condición de error.**
- Las **sentencias DML de lectura** realizan bloqueos compartidos y **no se ven afectadas por estos bloqueos**, pero utilizan la información **anterior a la transacción:**
  - Para ello, el SGBD debe mantener **la información anterior a cualquier transacción no terminada.**
  - Si el SGBD permite consultar información no confirmada con `COMMIT`, se dice que permite realizar *lectura sucia.*
  - Oracle no permite realizar *lectura sucia.*
- Si una **sentencia DML de modificación** de datos no puede obtener el bloqueo, **espera en una cola ordenada cronológicamente.**
- Esta espera se denomina **contención por bloqueo.**
  - La espera puede ser muy larga si hay muchas transacciones con muchas operaciones cada una.

Resolución de conflictos de bloqueo

- Se puede **controlar el tiempo de espera** utilizando una sentencia `SELECT` especial:

```
SELECT ... FOR UPDATE NOWAIT
SELECT ... FOR UPDATE WAIT n
```

- Esta sentencia es de lectura pero **bloquea como si se fueran a modificar datos.**
- que lo que hace es terminar inmediatamente (o después de `n` segundos) si las filas seleccionadas están bloqueadas por otra sesión.
- Cierto nivel de contención es normal en cualquier base de datos, pero **se puede deteriorar el rendimiento del sistema por errores de diseño o de programación de las aplicaciones.**

Contención por bloqueo

Problemas de diseño/programación que incrementan el nivel de contención:

- **Transacciones demasiado largas.** Ejemplo: si entre el bloqueo de una tabla y la sentencia `commit` se espera una acción del usuario (que está tomando café).
- **Procesos batch.** Ejemplo: el cierre mensual de la contabilidad: *conceptualmente* es una única transacción, pero en la práctica puede suponer el bloqueo de miles de filas de muchas tablas durante horas.
- **Aplicaciones externas.** Otras aplicaciones que utilizan Oracle pueden incluir niveles de contención muy altos.
- **Bloqueos para realizar consultas repetidas.** Algunas aplicaciones bloquean filas de la tabla para realizar varias consultas *consistentes* sobre una tabla. Se puede resolver con

  ```
  SET TRANSACTION READ ONLY
  ```

  No se bloquea ninguna fila pero garantiza que el estado de las tablas no está afectado por otras sesiones.

Interbloqueo

- Un tipo especial de bloqueo es el interbloqueo (**deadlock**)
- Se produce cuando dos transacciones se están esperando mutuamente.
- **Ejemplo:** dos transacciones ejecutando en dos sesiones distintas:

SESIÓN A

```
-- A Bloquea la cuenta 37
UPDATE Cuentas SET ...
WHERE Cta = '37';


-- A espera a B.
UPDATE Cuentas SET ...
WHERE Cta = '44';
```

SESIÓN B

```
-- B Bloquea la cta 44
UPDATE Cuentas SET ...
WHERE Cta = '44';


-- B espera a A.
UPDATE Cuentas SET ...
WHERE Cta = '37';
```

- Oracle lo detecta **y cancela automáticamente una de las transacciones lanzando una excepción.**

Control de Transacciones: **SET TRANSACTION**

- En una transacción, las filas de las tablas afectadas se van bloqueando **según se ejecuta la transacción:**
  - Si otras sesiones terminan transacciones con `COMMIT`, los cambios serán visibles **si no se han bloqueado antes las filas afectadas.**
- Esto puede ocasionar **inconsistencias en las transacciones.**
- Para evitarlo, se utiliza la sentencia **`SET TRANSACTION:`**

  `SET TRANSACTION [READ ONLY | READ WRITE] NAME` ***`nombre`***
  - **inicia una transacción explicitamente.**
  - **Todas las sentencias** que se incluyen en la transacción acceden a los datos en el estado en el que están en la BD **en el instante de inicio de la transacción.**
  - Las sentencias SQL **no ven los cambios realizados** por otros usuarios durante la ejecución de la transacción.
- La opción **`READ WRITE`** es la opción por defecto: fija el inicio de una transacción que modifica datos.
- La opción **`READ ONLY`** inicia una transacción **que no modifica datos.**
  - Se utiliza para que todas las consultas sean **consistentes con un estado de la BD:** el del inicio de la transacción.