UD5: MODELO FÍSICO DML Parte 2. Transacciones e Índices

Tema Extendido

Bases de Datos (BD)
CFGS DAM/DAW

Abelardo Martínez y Pau Miñana.
Basado y modificado de Sergio Badal y Raquel Torres.
Curso 2023-2024

ÍNDICE

1. Transacciones

Las transacciones son un aspecto crucial de los SGBD que garantizan la integridad, coherencia y fiabilidad de los datos. Una transacción en SQL representa una secuencia de sentencias DML (*INSERT*, *DELETE*, *UPDATE*) que se ejecutan como una única unidad de trabajo, es decir todas a la vez sin interferencias (por ejemplo de otros usuarios modificando las mismas tablas/datos). Cabe tener en cuenta que una transacción puede contener otras sentencias también, como DQL (*SELECT*) para consultar los datos.

Las propiedades fundamentales de una transacción, a menudo denominadas propiedades ACID, son Atomicidad, Consistencia, Aislamiento y Durabilidad.

- *Atomicidad*: Garantiza que una transacción se trate como una unidad de trabajo única e indivisible. O bien todos los cambios realizados por la transacción se consignan en la base de datos, o bien no se consigna ninguno. Si alguna parte de la transacción falla, toda la transacción vuelve a su estado anterior.

- *Consistencia*: Garantiza que una transacción lleve la base de datos de un estado válido a otro. Evita que se puedan romper las restricciones de integridad, deshaciendo los cambios en caso necesario.

- *Aislamiento*: Garantiza que la ejecución de una transacción esté aislada de la ejecución de otras transacciones. El aislamiento evita las interferencias entre transacciones garantizando la consistencia e integridad de los datos.

- *Durabilidad*: Garantiza que, una vez realizada una transacción, sus efectos sean permanentes y sobrevivan a cualquier fallo posterior, como una caída del sistema.

1.1. Inicio y fin de una transacción

Una transacción comienza con la primera instrucción DML que se ejecute o con la sentencia `START TRANSACTION;` y finaliza con alguna de estas circunstancias:

- Una operación `COMMIT;` o `ROLLBACK;`.
- Una instrucción DDL (como ALTER TABLE por ejemplo).
- Una instrucción DCL (como GRANT).
- El usuario abandona la sesión.
- Caída del sistema.

Hay que tener en cuenta que cualquier instrucción DDL, DCL, así como iniciar otra transacción con `START TRANSACTION;`, da lugar a un COMMIT implícito(automático).Es decir, todas las instrucciones DML ejecutadas hasta ese instante pasan a ser definitivas.

1.2. COMMIT

La instrucción `COMMIT;` confirma los cambios realizados por la transacción para que sean definitivos, irrevocables. Sólo se debe utilizar si estamos de acuerdo con los cambios, conviene asegurarse mucho antes de realizar el *COMMIT* ya que las instrucciones ejecutadas pueden afectar a miles de registros. Esto no es un repositorio, no se guardan varios estados, sólo el último.

> ⚠️ Mucho cuidado con los *COMMIT* implícitos por usar sentencias DDL, DCL o START TRANSACTION.

El cierre correcto de la sesión puede dar lugar a un COMMIT/ROLLBACK dependiendo del SGBD. Conviene ejecutar explícitamente la instrucción deseada para asegurarse. En MySQL no se ejecuta un *COMMIT* automático en el cierre de sesión, por ejemplo.

1.3. ROLLBACK

`ROLLBACK;` regresa la BD al punto anterior al inicio de la transacción, normalmente el último *COMMIT*, la última instrucción DDL/DCL o el inicio de la sesión.

Anula definitivamente los cambios, por lo que conviene también asegurarse de esta operación. Un abandono de sesión incorrecto o un problema de comunicación o una caída del sistema dan lugar a un *ROLLBACK* implícito.

1.4. Auto-commit

Algunos SGBD, como MySQL tienen un modo llamado autocommit que hace un COMMIT automático con cada sentencia DML. Por tanto las transacciones solo funcionan si se definen específicamente con `START TRANSACTION;` mientras que en otros SGBD son inherentes a las instrucciones DML.

El modo autocommit en MySQL empieza habilitado por defecto en cada sesión aunque se puede desactivar mediante `SET autocommit=0;`, de este modo las sentencias DML no se confirmarán automáticamente aunque no se inicie específicamente una transacción con `START TRANSACTION;` y funcionará como la mayoría de SGBD. Se puede volver a activar el funcionamiento con autocommit ejecutando `SET autocommit=1;`.

Los cambios en el funcionamiento del autocommit no afectan a la configuración del servidor, ni siquiera permanentemente a la del cliente; afectan sólo a la sesión actual, la siguiente vez que el cliente se conecte autocommit vuelve a estar activo.

Este modo de operación puede ser desconcertante si se tiene experiencia con otros sistemas de bases de datos, donde es una práctica estándar emitir una secuencia de instrucciones DML y confirmarlas (*COMMIT*) o deshacerlas (*ROLLBACK*) todas juntas.

1.5. Estado de los datos durante la transacción

Recordemos que una transacción se inicia con un comando DML o START TRANSACTION. Cuando esto sucede, hay que tener en cuenta que:

- Se puede volver a la instrucción anterior al inicio de la transacción cuando se desee (*ROLLBACK*).
- Las instrucciones de consulta (*SELECT*) realizadas por el usuario que inició la transacción muestran los datos ya modificados por las instrucciones DML (*INSERT*, *DELETE*, *UPDATE*).
- El resto de usuarios, en cambio, ven los datos tal cual estaban antes de la transacción, de hecho los registros afectados por la transacción aparecen bloqueados hasta que la transacción finalice. Esos usuarios no podrán modificar los valores de dichos registros.
- Tras la transacción, todos los usuarios ven los datos tal cual quedan tras el fin de transacción. Los bloqueos son liberados y los puntos de ruptura borrados.

1.6. Ejemplos

Ejemplo 1: Transacción Simple

Usaremos la tabla de departamentos de las unidades anteriores, borraremos todos los departamentos anteriores y crearemos uno nuevo. Posteriormente desharemos los cambios:

```
START TRANSACTION;
DELETE FROM departamentos;
INSERT INTO departamentos (cod_dpt, nombre_dpt, ubicacion) VALUES
('MKT', 'Márqueting', 'Planta segunda U2');
SELECT * FROM departamentos;

-- Anulamos los cambios
ROLLBACK;
SELECT * FROM departamentos;
```

```
mysql> SELECT * FROM departamentos;
+---------+-------------+-------------------+
| cod_dpt | nombre_dpt  | ubicacion         |
+---------+-------------+-------------------+
| MKT     | Márqueting  | Planta segunda U2 |
+---------+-------------+-------------------+
1 row in set (0,00 sec)
mysql> ROLLBACK;
Query OK, 0 rows affected (0,01 sec)
mysql> SELECT * FROM departamentos;
+---------+-----------------+-------------------+
| cod_dpt | nombre_dpt      | ubicacion         |
+---------+-----------------+-------------------+
| ADM     | Administración  | Planta quinta U2  |
| ALM     | Almacén         | Planta baja U1    |
| COM     | Comercial       | Planta tercera U3 |
| CONT    | Contabilidad    | Planta quinta U1  |
| INF     | Informática     | Planta sótano U3  |
+---------+-----------------+-------------------+
```

Con un COMMIT en lugar del ROLLBACK se hubiesen confirmado los cambios.

Ejemplo 2: Autocommit

Comprobamos ahora el funcionamiento del autocommit en MySQL. Primero comprobamos que el funcionamiento por defecto es con autocommit activado (las sentencias DML se auto-confirman y no se pueden deshacer)

```sql
INSERT INTO departamentos (cod_dpt, nombre_dpt, ubicacion) VALUES
('MKT', 'Márqueting', 'Planta segunda U2');
ROLLBACK;
-- Autocommit hace que el ROLLBACK no sea aplicable.
SELECT * FROM departamentos;
```

```
mysql> INSERT INTO departamentos (cod_dpt, nombre_dpt, ubicacion)
VALUES -> ('MKT', 'Márqueting', 'Planta segunda U2');
Query OK, 1 row affected (0,01 sec)
mysql> ROLLBACK;
Query OK, 0 rows affected (0,00 sec)
mysql> SELECT * FROM departamentos;
+---------+-----------------+-------------------+
| cod_dpt | nombre_dpt      | ubicacion         |
+---------+-----------------+-------------------+
| ADM     | Administración  | Planta quinta U2  |
| ALM     | Almacén         | Planta baja U1    |
| COM     | Comercial       | Planta tercera U3 |
| CONT    | Contabilidad    | Planta quinta U1  |
| INF     | Informática     | Planta sótano U3  |
| MKT     | Márqueting      | Planta segunda U2 |
+---------+-----------------+-------------------+
```

Lo borramos, desactivamos el autocommit y repetimos el procedimiento anterior

```
--Borramos 'MKT'
DELETE FROM departamentos WHERE cod_dpt='MKT';
SELECT * FROM departamentos;
-- Desactivamos autocommit, ahora todo DML necesita confirmación
SET autocommit=0;
INSERT INTO departamentos (cod_dpt, nombre_dpt, ubicacion) VALUES
('MKT', 'Márqueting', 'Planta segunda U2');
ROLLBACK;
-- Rollback deshace el Insert, estamos en una transacción.
SELECT * FROM departamentos;
-- Reactivamos autocommit
SET autocommit=1;
```

```
mysql> SET autocommit=0;
Query OK, 0 rows affected (0,00 sec)
mysql> INSERT INTO departamentos (cod_dpt, nombre_dpt, ubicacion)
VALUES -> ('MKT', 'Márqueting', 'Planta segunda U2');
Query OK, 1 row affected (0,00 sec)
mysql> ROLLBACK;
Query OK, 0 rows affected (0,01 sec)
mysql> SELECT * FROM departamentos;
+---------+----------------+-------------------+
| cod_dpt | nombre_dpt     | ubicacion         |
+---------+----------------+-------------------+
| ADM     | Administración | Planta quinta U2  |
| ALM     | Almacén        | Planta baja U1    |
| COM     | Comercial      | Planta tercera U3 |
| CONT    | Contabilidad   | Planta quinta U1  |
| INF     | Informática    | Planta sótano U3  |
+---------+----------------+-------------------+
```

En muchos SGBD el funcionamiento por defecto es así, no hay autocommit y no se necesita una sentencia específica para iniciar la transacción. No es el caso de MySQL como hemos visto.

Ejemplo 3: Conflictos entre usuarios

Para este ejemplo se necesitan 2 clientes conectados a la misma base de datos, con 2 usuarios distintos. El tratamiento de usuarios lo veremos en unidades posteriores.

Cliente 1

Se cambia el código de 'INF' a 'IT'.

```
START TRANSACTION
UPDATE departamentos SET cod_dpt='IT'
WHERE cod_dpt='INF';
-- Se ejecuta ahora un SELECT en el cliente2 antes del COMMIT
COMMIT;
```

Cliente 2

Antes de confirmar la transacción en el cliente 1 consultamos los datos en el cliente 2. El código sigue mostrándose como 'INF'. Si se intentase un UPDATE sobre la tabla este se quedaría bloqueado hasta que el cliente 1 confirme o anule la transacción.

```
SELECT * FROM departamentos;
```

Si se vuelve a ejecutar tras el COMMIT los cambios ya serán visibles. Y se podrán modificar los datos. Esto se usa para evitar que se apliquen cambios simultáneos entre varios clientes que entren en conflicto entre ellos.

2. Índices

> 📝 Los índices son un objeto más de las bases de datos, como pueden ser las tablas, que hacen que las bases de datos aceleren las operaciones de consulta y ordenación de los datos (DQL o SELECT).

De cada tabla, se puede crear uno o varios índices, que funcionan como una copia de esa tabla ordenada en función de un campo o criterio concreto (pudiendo implicar varios campos). Por ejemplo, se puede tener una tabla de *ESTUDIANTES(DNI, nombre, apellidos, f_nacimiento)* con un índice sobre el campo *f_nacimiento* que permita consultar rápidamente todos los estudiantes ordenados por ese campo.

Se almacenan aparte de la tabla a la que hace referencia, lo que permite crearlos y borrarlos en cualquier momento. No sólo agilizan las búsquedas ordenadas sino cualquier búsqueda que filtre usando el campo o los campos en el índice.

La mayoría de los índices se crean de manera implícita (automática), como consecuencia de las restricciones PRIMARY KEY, UNIQUE y FOREIGN KEY. Estos son índices obligatorios, por los que los crea el propio SGBD.

Por ejemplo, si creamos la tabla ESTUDIANTES con {nombre,apellidos} como UK y un id_ciudad de residencia como FK estamos, implícitamente, creando estos objetos en la base de datos:

- La tabla *ESTUDIANTES*
- Una estructura indexada ordenada por la PK *DNI*.
- Una estructura indexada ordenada por la FK *id_ciudad*.
- Una estructura indexada ordenada por la UK *{nombre, apellidos}*.

De esta manera, cada vez que hagamos un *DML* (INSERT, UPDATE, DELETE) sobre esa tabla tendremos que reconstruir todos esos objetos por lo que **los índices son recomendados solo cuando las operaciones DML son infinitamente menores que las de consulta** (*DQL* o *SELECT*).

Por tanto:

- Los índices realizan una lista ordenada por la que el SGBD puede acceder para facilitar la búsqueda y ordenación de los datos. Ayuda a acelerar ciertas consultas (*SELECT*).
- Cada índice añade una nueva estructura, con lo que las instrucciones DML se tienen que aplicar sobre éstas y reordenarse, con lo que se ralentizan notablemente.
- Se gana en velocidad de consulta a costa de almacenamiento y velocidad en la modificación e inserción de datos, estos deben ser los criterios para elegir cuando crear un índice o no.

2.1. Creación de índices

Aparte de los índices implícitos (automáticos) comentados anteriormente, se pueden crear índices de forma explícita (manual). Éstos se crean para aquellos campos sobre los cuales se realizarán búsquedas e instrucciones de ordenación frecuente.

```
CREATE INDEX nombre ON [esquema.]nombre_tabla
({nombre_columna | (expr)} [ASC | DESC] [,{nombre_columna...}]...)
```

Simplificando:

```
CREATE INDEX nombre
ON tabla (columna1 [,columna2...])
```

Ejemplo:

```
CREATE INDEX est_nom_ix
ON estudiantes (apellidos, nombre);
```

El ejemplo crea un índice para los campos apellidos y nombre en conjunto, es decir una estructura sobre la tabla ordenada alfabéticamente por los apellidos y en caso ser iguales por el nombre de los estudiantes. Por tanto este índice es efectivo solo cuando se buscan y ordenan estudiantes usando ese criterio. Si realizamos una consulta sobre esa tabla ordenada por esos campos y/o en ese orden, la respuesta será de milisegundos aunque la tabla tenga millones de registros.

Se aconseja crear índices en campos que:

- Contengan una gran cantidad de valores repetidos.
- Contengan una gran cantidad de nulos.
- Sean parte habitual de cláusulas *WHERE*, *GROUP BY* u *ORDER BY* (que veremos más adelante).
- Sean parte de listados de consultas de grandes tablas sobre las que casi siempre se muestran como mucho un 4% de su contenido.

No se aconseja en campos que:

- Pertenezcan a tablas pequeñas.
- No se usen a menudo en las consultas.
- Pertenezcan a tablas cuyas consultas muestren más de un 4% del total de registros.
- Pertenezcan a tablas que se actualicen frecuentemente (*DML*).

2.2. Lista de índices

Lo organización de los índices es un asunto propio de cada SGBD, con lo que no existe una nomenclatura estándar para consultarlos. Se suele almacenar la información en algunas tablas/vistas que se pueden consultar, pero los nombres son distintos en cada uno

Para ver la lista de índices en Oracle se utiliza la vista *USER_INDEXES*. Mientras que la vista *USER_IND_COLUMNS* muestra la lista de columnas que son utilizadas por índices.

> 📝 En MySQL se usa el comando `SHOW INDEX FROM nombre_tabla;` que nos devolverá la información de la tabla de índices referente a esa tabla. Se puede filtrar con un WHERE.

2.3 Borrar índices

La instrucción `DROP INDEX nombre_indice;` permite eliminar el índice en cuestión.

3. Bibliografía

- MySQL 8.0 Reference Manual. https://dev.mysql.com/doc/refman/8.0/en/
- Oracle Database Documentation https://docs.oracle.com/en/database/oracle/oracle-database/index.html
- W3Schools. MySQL Tutorial. https://www.w3schools.com/mysql/
- GURU99. Tutorial de MySQL para principiantes Aprende en 7 días. https://guru99.es/sql/
- SQL Tutorial - Learn SQL. https://www.sqltutorial.net/