ArgentinaPrograma
YoPrograma

Transacciones, vistas, índices y disparadores

por Leonardo Blautzik, Federico Gasior y Lucas Videla

Julio / Diciembre 2021

Ministerio de
Desarrollo Productivo
Argentina

Objetos del motor de base de datos

Una base de datos esta compuesta por distintos objetos con distintas funcionalidades

Hasta ahora vimos objetos relacionales que son tablas, filas y columnas

En esta clase conoceremos muchos más y algunos los veremos en profundidad

Objetos del motor de base de datos

Los objetos más comunes son:

- Relacionales (tablas, columnas, filas)
- Transacciones
- Vistas
- Indices
- Disparadores
- Cursores
- Particiones
- Procedimientos almacenados

Transacción

Unidad de trabajo realizado dentro del sistema de administración de base de datos contra la base de datos

Generalmente representa un cambio en la base de datos

Se utilizan principalmente para contener un trabajo y permitir que el mismo y la base de datos se recuperen correctamente a pesar de cualquier problema, y para proveer aislamiento contra otros accesos concurrentes a la base de datos

Transacciones

Por definición deben ser ACID:

- Atómicas
- Consistentes
- Aisladas (Isolated)
- Duraderas

Transacciones en SQL

Comandos

- Para comenzar una transacción se debe usar START TRANSACTION (o BEGIN)
- Para cerrar la transacción y guardar los cambios se debe usar COMMIT
- ROLLBACK también finaliza la transacción pero descarta todos los cambios

El comportamiento de que ocurre una vez iniciada una transacción puede variar entre motores de bases de datos y conexiones o modos de las mismas. Por ejemplo el comportamiento al tener un error en una consulta puede ser el de ignorarlo, o el de hacer un ROLLBACK implícito

Transacciones en SQLite

SQLite funciona con un modo llamado **autocommit** por defecto. Este modo hace que por cada comando ejecutado, SQLite inicie, proceses y finalice una transacción automáticamente

Si se desea hacer manualmente, además de desactivar este modo, se debe iniciarla con BEGIN o BEGIN TRANSACTION

La transacción unicamente finalizará al ejecutarse el comando COMMIT o ROLLBACK (puede ser de manera implícita)

Solo se permite una transacción a la vez, salvo que se utilice SAVEPOINT

Ejemplo

Se ejecutará un débito de dos cuentas distintas a la vez para realizar un pago de 4500 en partes de 2000 en la cuenta 1542 y el restante en la 79. Los punto y coma (**;**) son muy importantes para separar sentencias

```
BEGIN TRANSACTION;

UPDATE cuenta SET saldo = saldo - 2000
WHERE id = 1542;
UPDATE cuenta SET saldo = saldo - 2500
WHERE id = 79;

COMMIT;
```

¿Transacción incorrecta?

Si en el ejemplo no hubiese tenido saldo en la segunda cuenta, hubiese restado correctamente de la primera a pesar del error en la segunda linea

Esto ocurre por como se aplica el manejo de errores implícito. Si quisiera volver los cambios atrás y no hacer el commit, debería decir explícitamente que si hay error al actualizar, realice un rollback

```
UPDATE OR ROLLBACK cuenta
SET saldo = saldo - 2500
WHERE id = 79;
```

Nota: Saldo debería tener un CHECK que no permita que su valor sea negativo, sino la consulta no fallará

Conflictos

SQLite tiene 5 maneras de manejar conflictos

- ROLLBACK
- ABORT (defecto y estandar)
- FAIL
- IGNORE
- REPLACE (UNIQUE o PRIMARY KEY)

Ninguna de las mencionadas funciona con las restricciones de FOREIGN KEY

Conflictos

Los conflictos se pueden declarar en la mayoria de restricciones a través de ON CONFLICT

También pueden declararse directamente en consultas de INSERT o UPDATE

Trabajo en clase

Transacciones

Vistas

Es el resultado de una consulta que fue almacenada en forma de tabla

Al contrario de una tabla ordinaria, una vista no tiene datos almacenados por si misma

La consulta vuelve a ejecutarse cada vez que se quiere consultar la información de la vista

Ventajas

Algunas ventajas de las vistas sobre las tablas son

- Dar contexto o permiso sobre una porción de una tabla
- Relacionar contenidos de multiples tablas
- Ocupar muy poco espacio
- Relacionar a información distinta dependiendo de un contexto
- Realizar operaciones de escritura dentro de los datos de la propia vista

Vistas en SQLite

Lamentablemente solo disponemos de vistas de solo lectura, por lo tanto operaciones como INSERT, UPDATE y DELETE sobre las vistas no están permitidas.

Las vistas pueden crearse temporales, lo cual hará que solo existan hasta que la conexión con la base de datos sea cerrada

Pueden ser creadas sin especificar explícitamente las columnas, solo con la consulta (como también ocurre con las tablas creadas con consultas)

Ejemplo

```
CREATE VIEW ventas_por_sucursal AS
SELECT sucursales.nombre, count(*) AS 'cantidad ventas'
FROM ventas
JOIN vendedores ON vendedores.id = ventas.id_vendedor
JOIN sucursales ON sucursales.id = vendedores.sucursal
GROUP BY sucursales.id
```

Trabajo en clase

Vistas

Índices

Son estructuras de datos pensadas para mejorar la velocidad de consultas de datos de tablas

Deben sacrificar tiempo de escritura y espacio de almacenamiento

Deben construirse sabiamente ya que un mal índice probablemente no sea usado, pero sus desventajas igual aplican

Índices

Pueden ser parciales o totales

Pueden ser o no ser únicos

Pueden incluir una o varias columnas (de la misma tabla)

Pueden ser expresiones (como operaciones matemáticas)

Ejemplo

```
CREATE INDEX "anual" ON "ventas" (
  strftime('%Y', fecha)
);

SELECT *
FROM ventas
WHERE strftime('%Y', fecha) > 2010
```

Para saber que índices son usados, se puede utilizar EXPLAIN QUERY PLAN antes del SELECT

Detalles de índices en SQLite

Se puede forzar el uso de un índice en una consulta a través del INDEXED BY. Con esto se pueden probar distintos indices y medir su eficacia

Si se utilizan índices con expresiones y se modifican sus definiciones, o se modifica la codificación de caracteres, es necesario invocar a REINDEX para volver a crear los índices y corregir fallos provocados por estos cambios

No se puede ordenar directamente por NULLS o NOT NULLS, pero a los NULLS se los considera como menores a cualquier otro valor

Trabajo en clase

Índices

Disparadores (triggers)

Es un objeto de la base de datos que es ejecutado automáticamente cuando se realizan acciones tales como INSERT, UPDATE y DELETE

Cada disparador se puede asociar a un momento de una acción de una tabla

Son utilizados para manejar reglas complejas de negocio que no pueden ser manejadas de otra manera dentro de la base de datos

Tipos de disparadores

Combinando todas las posibilidades de asociación tenemos:

- BEFORE INSERT
- AFTER INSERT
- BEFORE UPDATE
- AFTER UPDATE
- BEFORE DELETE
- AFTER DELETE

Referencia hacia la tupla original

INSERT: NEW hace referencia a la tupla a ser insertada

UPDATE: NEW hace referencia a la tupla a ser actualizada y OLD a la tupla que va a ser reemplazada

DELETE: OLD hace referencia a la tupla a ser borrada

Ejemplo

La siguiente consulta muestra como crear un disparador en la tabla usuarios, para crear una cuenta por defecto al crear un usuario

```
CREATE TRIGGER crear_cuenta_al_crear_usuario
  AFTER INSERT ON usuarios
BEGIN
  INSERT INTO cuentas (usuario_id, fecha_creacion)
  VALUES (NEW.id, datetime('now'));
END;
```

Trabajo en clase

Disparadores

INSTEAD OF

Además de BEFORE y AFTER como modificador del disparador tenemos INSTEAD OF

Permite modificar el comportamiento de INSERT, UPDATE y DELETE de una tabla o vista

Puntualmente en las vistas, permite ejecutar un disparador cuando se quiere realizar un INSERT, UPDATE o DELETE, lo cual hace que estos comandos puedan ser ejecutados efectivamente en una vista

RAISE()

Permite lanzar un comando de un tipo de conflicto para ser tomado por un ON CONFLICT

Se pueden utilizar los tipos de conflicto IGNORE, ROLLBACK, ABORT y FAIL junto con un mensaje de error

¡Muchas Gracias!

continuará...

Ministerio de
Desarrollo Productivo
Argentina