Trabajo Práctico 7

Integridad y Transacciones

- **Alumno**: Sussini Guanziroli, Patricio
- **Materia**: Bases de datos I
- **Tutora**: Constanza Uño

Parte 1: Fundamentos de Integridad y Consistencia

1. La integridad de los datos hace referencia a la precisión, coherencia y confiabilidad de los datos a lo largo de su ciclo de vida.

   Tipos de integridad:

   - **Integridad de Entidad**: Cada fila siendo única. Ej: DNI de una persona.
   - **Integridad Referencial**: Mantiene las relaciones funcionales entre las tablas. Ej: Un pedido no puede ser de un cliente NULL.
   - **Integridad de Dominio:** Garantiza que los valores sean siempre válidos. Ej: El precio de un producto siempre debe ser mayor a 0.

2. Concepto de **consistencia**: La consistencia asegura que la base de datos siempre pasa de un estado válido a otro, sin “crear” o “destruir” valores. Ejemplo **transferencia**: Dentro de la transferencia, el estado inicial es válido, con un total definido y el estado final también. La consistencia se rompe si el dinero se debita de una cuenta pero no se acredita en la otra, o viceversa.
3. La **disponibilidad** es la capacidad de la base de datos para estar siempre accesible y funcionando para los usuarios, incluso cuando múltiples la estan accediendo en simultáneo.
   - **Aspectos Clave:**
     i. Tiempos de actividad y confiabilidad del sistema.
     ii. Tiempo de respuesta bajo carga (uso).
     iii. Capacidad para manejar acceso concurrente.
     iv. Recuperación ante fallas o desperfectos técnicos.

Parte 2: Transacciones y Propiedades *ACID*

1. Una **transacción** es una *unidad lógica* de trabajo que contiene una o más operaciones de base de datos (lecturas/escrituras).

   Características clave:

   a. Se ejecuta como UNA sola unidad indivisible (A).
   b. Todas las operaciones se completan con éxito (se commitean) o ninguna lo hace (reculada).
   c. Mantiene la consistencia de la base de datos.
   d. Aísla las operaciones múltiples concurridas.

2. Propiedades **ACID:**

   a. Atomicidad:

      i. Todas las operaciones tienen éxito o fallan. No hay transacciones parciales, es decir, no se pueden dividir.
      ii. Ej: Una transferencia, primero resta saldo como una transacción y luego “otra” lo acredita.

   b. Durabilidad:

      i. Una vez concretada una transacción, sus efectos persisten inclusive a través de fallas del sistema.
      ii. Los cambios se almacenan permanentemente en unidades de almacenamiento permanente, es decir, NO volátiles.
      iii. Ej: Se escribe en una memoria volátil o no se realizan copias de seguridad.

   c. Consistencia:

      i. La transacción independientemente transforma la base de datos de un estado válido directamente hacia otro estado válido.
      ii. Se aplican la totalidad de reglas, restricciones y disparadores, que aseguran dicha consistencia.
      iii. Ej: Se escriben datos en estado no válido, es decir, stock en negativo.

   d. Aislamiento:

      i. Las transacciones concurrentes se ejecutan como si fueran secuenciales.

ii. Ej: Se deja ver el saldo de una cuenta que debe ser disminuído antes de sustraer, entonces se permite otra transacción de retiro con saldo fantasma.
iii. Una transacción no puede ver los estados en el medio de otra transacción, hasta que esta se confirme y el nuevo estado quede escrito (commit).

3. Sentencias de transacción**:**
    a. **BEGIN** (comenzar): Esta sentencia es el punto de partida. Se comunica con la base de datos dando aviso de que comienza una transacción, anotando los cambios pero sin cambiar nada todavía.
    b. COMMIT (confirmación): Esta sentencia es de confirmación. Se ejecuta cuando todas las operaciones dentro del bloque de la transacción se han completado sin errores. Al ejecutar la sentencia la base de datos recibe la alta de que se puede proseguir con los cambios para que desde el BEGIN se escriban y estén disponibles para todos los usuarios.
    c. ROLLBACK (reculada): Se puede tomar en cuenta como una “salida de emergencia”. Si ocurre algún error durante la transacción, por ejemplo: si se intenta debitar dinero de una cuenta sin fondos o el sistema se cae, la sentencia le comunica a la base de datos que no se puede realizar dicha transacción y se le indica deshacer todo desde el BEGIN. La base de datos entonces vuelve todo para atrás de la forma en que se encontraba previo al inicio.

    Estas tres sentencias securizan la atomicidad, volviendo capsula todas las modificaciones, asegurando que no sufrirán cambios a menos que todo salga bien.

Parte 3: Control de Concurrencia

1. La **concurrencia** se refiere a múltiples usuarios o procesos que acceden y potencialmente modifican los mismos datos de manera simultánea.
   Desafíos principales:
   - Mantener la integridad de los datos.
   - Evitar interferencias entre transacciones.
   - Equilibrar aislamiento y restricción con rendimiento.
   - Evitar bloqueos y problemas de desenvoltura.
2. Existen 4 problemas más comunes y recurrentes para la **concurrencia:**
   a. **Lecturas sucias:** Es cuando una transacción lee datos que han sido modificados por otra transacción pero que todavía no han sido confirmados por la misma. Por lo que se juega la chance de lectura de datos no válidos o erróneos.
   b. **Lecturas no repetibles:** Una transacción lee los mismos datos dos veces pero obtiene resultados diferentes, porque otra transacción modificó y confirmó los datos públicos de la db entre las dos lecturas.
   c. **Lecturas fantasma:** Una transacción atomizada ejecuta una consulta dos veces (lectura) y obtiene un conjunto de filas diferente en cada iteración, dado que otra transacción ha modificado (insertado o eliminado) filas que coinciden con la condición de la consulta.
   d. **Actualizaciones perdidas:** Una transacción lee y actualiza datos y otra transacción diferente actualiza y lee los mismos datos, y una de las dos actualiza y confirma cambios sobre la marcha sin incorporarlos, lo que causa una actualización totalmente perdida y olvidada.
3. Niveles de aislamiento:
   a. **Explicación individual:**
      i. **READ UNCOMMITED:** Permite que una transacción lea los cambios hechos por otra transacción inclusive si esos cambios no han sido confirmados y guardados. Es la que mejor rendimiento tiene porque no hay esperas, pero permite todos los problemas. Lecturas sucias, No repetibles, fantasma y pérdida de actualizaciones.

ii. **READ COMMITED:** O en su traducción “Lectura confirmada”, admite que las transacciones solo lean los cambios que han sido confirmados finalmente por otras, lo que evita las lecturas intermedias no atomizadas. **Previene** problemas de lecturas sucias, pero aún permite lecturas no repetibles, y lecturas fantasma. El rendimiento sigue siendo alto, por lo que es estándar en aplicaciones dado su buen equilibrio.

iii. **REPEATABLE READ:** Traducido como “Lectura repetible”, garantiza que si lees el mismo dato varias veces dentro de una misma transacción, siempre obtendrás el mismo resultado. Siempre la base de datos tiene una “foto” de los datos al comienzo de tu transacción.

Previene problemas de Lecturas sucias y como su nombre indica, de lecturas no repetibles. Aún sigue permitiendo las lecturas fantasma, si la “foto” se queda desactualizada por una transacción posterior.

El rendimiento es medio, y constituye el nivel por defecto de MySQL, aporta un alto grado de seguridad sin sacrificar demasiado rendimiento.

iv. **SERIALIZABLE:** Es el nivel más estricto de seguridad y restricción. En síntesis fuerza a las transacciones a ejecutarse en fila, teniendo una tras otra que esperar que la anterior termine de leer y escribir.

Previene todos los problemas al no permitir más de una transacción a la vez.

El rendimiento se ve fuertemente afectado si hay muchas consultas en simultáneo.

b. El nivel de aislamiento tiene una fuerte correlación con el rendimiento, ya fue explicado por caso particular a nivel en cada ítem anterior.

Hay que tener en cuenta el propósito de la base de datos, evaluando la sensibilidad de los datos con los que se trabaja para determinar si es o no justificable sacrificar rendimiento por protección.

c. **Sintaxis en SQL:**

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;
-- Agrego sentencias SQL acá
COMMIT;
```

4. Mecanismos diversos de Control de Concurrencia:
   a. **Bloqueo:** Es el mecanismo más básico, cuando una transacción necesita usar un dato, la primera le pone un bloqueo para que las demás no puedan interferir (leerlo o escribirlo).
      i. **Bloqueo compartido:** Es un bloqueo de solo lectura, todas las transacciones puede leerlo pero solo la primera escribirlo.
      ii. **Bloqueo exclusivo:** Candado de escritura. Solo una transacción puede tener ese bloqueo. Impide que las demás lean y escriban ese dato.
   b. **Ordenamiento por marca de tiempo:** En lugar de bloquear se asignan marcas de tiempo a las transacciones y ejecuta las operaciones en función del orden. Esto previene conflictos y previene inconsistencias.
   c. **Control de Concurrencia con Múltiples Versiones:** Este es el método por defecto en MySQL. Funciona de manera de que cuando una transacción comienza obtiene una “instantánea” de cómo se veían los datos en ese momento, por ende mantiene varias versiones en simultáneo. Permite operaciones de lectura sin bloqueo. Esto mejora enormemente la velocidad.

5. Un **bloqueo mutuo** o deadlock es un círculo vicioso que ocurre cuando dos o más transacciones se quedan atascadas, cada una esperando que la otra libere un recurso que necesita para continuar. Esto genera un punto muerto donde ninguna puede seguir.

   Para ser resuelto, se aplican algoritmos de detección de bloqueos, lo que elige una de las dos transacciones para ser abortada y terminada. La “víctima” recula y libera el recurso para que la otra termine y siga todo su normal curso.

Parte 4: Aplicaciones en el Mundo Real

1. Tres aplicaciones reales donde las transacciones con críticas para mantener integridad y consistencia de datos:
    a. **MercadoPago:** Todos conocemos y usamos la billetera virtual del gigante tecnológico bonaerense. En muchos casos transacciones con información muy sensible son realizadas bajo condiciones mucho menos que ideales, como internet inestable, clientes sin alimentación contínua o intentos de fraude constantes. MercadoPago tiene que asegurar que las transacciones siempre sean perfectamente efectuadas, ya que en muchos casos pueden fallar, y debe rápidamente ser solucionada por el sistema.
    b. **Flybondi:** La aerolínea low-cost nacional, tiene un sistema de reservas online, donde los clientes eligen y personalizan la totalidad de la información para su vuelo. Es de suma importancia tener extrema protección de datos para no vender el mismo asiento dos veces o reservar asientos que luego nadie va a usar. De esa forma tienen que contar con extensas normas de aislamiento y seguridad para concurrencia extrema y limitada.
    c. **PedidosYA:** La aplicación de pedidos tiene su **propio almacén** de productos con su stock y disponibilidad rigurosamente controlado. Es por eso que no se pueden dar el lujo de vender productos que no tienen o vender el mismo producto a dos clientes diferentes. Al ser una base de datos muy concurrida es importante generar controles de concurrencia y transacciones bien atomizadas.

2. Investigación:
    a. Estrategias diferentes para manejar los problemas de concurrencia:

Control de concurrencia Optimista vs Pesimista:

Son dos enfoques diferentes para resolver el mismo problema. ¿Cómo evitar que los usuarios se pisen entre sí para modificar los mismos datos? La elección depende de la probabilidad de conflicto que tengamos.

- **Control Pesimista:** Es la estrategia de bloqueo tradicional, cuando los conflictos son MUY probables.
    - Funciona bloqueando primero (pidiendo permiso) hasta que la transacción que bloquea libere el dato y así continuar.
    - Es ideal para los entornos de alta contención, donde es muy probable que muchos usuarios accedan a la misma información al mismo tiempo.
    - Ej: Sistemas de reserva de FlyBondi. Conciertos en Ticketek.
- **Control Optimista:** Es una estrategia que asume que los conflictos son muy poco probables.
    - Funciona bajo la lógica de “acceder y después ver”.
    - Un usuario lee un dato pero no lo bloquea. La base de datos permite que otros usuarios también accedan y lo modifiquen. Cuando el primero intenta guardar sus cambios la base de datos verifica si el dato original ha sido modificado por alguien más desde que él lo leyó.
    - Si hubo algún cambio, la actualización del usuario falla y debe reintentar la operación.
    - Es perfecto para entornos de baja concurrencia donde los conflictos son raros, y tiene mejor rendimiento y escalabilidad.
    - Ej: Una página de comentarios de un blog. La edición de artículos en WikiPedia.

Fuentes utilizadas:

- https://learn.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-locking-and-row-versioning-guide?view=sql-server-ver17
- https://aws.amazon.com/blogs/database/implement-resource-counters-with-amazon-dynamodb/

b. Implementación de ACID en MySQL.

Teniendo en cuenta el SGBD de MySQL, usando InnoDB como motor.

A. **Atomicidad:** InnoDB garantiza la atomicidad a través del **undo log.** Antes de modificar cualquier dato, InnoDB escribe la versión antigua de ese dato en el log. Si la transacción falla y hace falta una reculada, el sistema simplemente usa esta bitácora para revertir los cambios.

- C. **Consistencia:** Se logra mediante la combinación de mecanismos. Al iniciar una transacción, el sistema está consistente. InnoDB asegura que se cumplan todas las restricciones. Si una transacción violara una de estas reglas, la atomicidad vía reculada se encarga de devolver los datos a un estado anterior.
- **I. Aislamiento:** Se usa el Control de Concurrencia Multiversión, Como se vió previamente, esto permite que cada transacción trabaje sobre una “instantánea” de los datos, permitiendo que las lecturas y escrituras no se bloqueen entre sí. También usa bloqueos.
- D. **Durabilidad:** La misma se garantiza mediante el **log redo.** Cuando se confirma una transacción, los cambios se escriben en forma secuencial como primera instancia y muy rápido en el log del disco. Si el servidor se apaga antes el InnoDB puede reproducir el **redo log** para recuperar todas las transacciones confirmadas y asegurar que no se pierda ningún dato.

Niveles de aislamiento:

- Por defecto en MySQL el motor tiene el nivel de aislamiento REPETABLE READ.

Fuentes de referencia:

- https://dev.mysql.com/doc/refman/8.0/en/mysql-acid.html
- https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html
- https://dev.mysql.com/doc/refman/8.4/en/innodb-locking-transaction-model.html