TECNICATURA UNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA

TP 7 - Unidad 7
Integridad y Transacciones

Nombre: Farias, Gustavo
Comisión: M2025-13
Matrícula: 101662
Repositorio GitHub:
https://github.com/Lucenear/UTN-TUPaD-TPs/tree/main/Bases%20de%20Datos/Bases%20de%20Datos%20I

Parte 1: Fundamentos de la Integridad y Consistencia

1. Defina qué es la integridad de los datos y explique por qué es crucial. Mencione los tres tipos de restricciones de integridad con un ejemplo para cada una.

La integridad de los datos se refiere a la precisión, confiabilidad y consistencia de los datos a lo largo de su vida útil. Es crucial porque garantiza que la información almacenada sea válida y confiable, lo que es fundamental para que las aplicaciones y los sistemas que dependen de ella funcionen correctamente y tomen decisiones acertadas.

Los tres tipos principales de restricciones son:

- Integridad de Entidad: Asegura que cada fila en una tabla sea única e identificable.
    - Ejemplo (MySQL): Una tabla Usuarios con una columna id_usuario como PRIMARY KEY AUTO_INCREMENT. Esto impide que haya dos usuarios con el mismo ID y que el ID sea nulo.

- Integridad Referencial: Mantiene las relaciones consistentes entre tablas. Una clave foránea en una tabla debe hacer referencia a una clave primaria existente en otra.
    - Ejemplo (MySQL): Una tabla Pedidos con una columna id_cliente definida como FOREIGN KEY REFERENCES Clientes(id_cliente). Esto impide que se pueda asignar un pedido a un cliente que no existe.

- Integridad de Dominio: Asegura que los valores de una columna cumplan con un formato, tipo de dato o rango específico.
    - Ejemplo (MySQL): Una tabla Productos con una columna precio definida como DECIMAL(10,2) CHECK (precio > 0). Esto garantiza que el precio sea un número positivo y con dos decimales.

2. Utilizando el ejemplo de la transferencia bancaria, explique el concepto de consistencia de la base de datos.

La consistencia garantiza que una base de datos pase de un estado válido a otro estado válido después de una operación, cumpliendo todas las reglas definidas.

En el ejemplo de la transferencia bancaria de $100 de la Cuenta A a la Cuenta B:

- Estado inicial válido: Cuenta A = $1000, Cuenta B = $500. Total del sistema = $1500
- Operación:
    - Se resta $100 a la Cuenta A (nuevo saldo: $900)
    - Se suma $100 a la Cuenta B (nuevo saldo: $600)
- Estado final válido: Cuenta A = $900, Cuenta B = $600. Total del sistema = $1500

La consistencia asegura que ambos pasos se completen. Si después del paso 1 falla el paso 2 (por ejemplo, la Cuenta B no existe), el sistema deshace (ROLLBACK) el primer paso. Así, el dinero no "desaparece" y la base de datos se mantiene en un estado válido (el inicial), preservando la regla de negocio de que el dinero total se conserva.

3. ¿Qué se entiende por disponibilidad de la base de datos? Describa los aspectos clave.

La disponibilidad es la capacidad de una base de datos para permanecer accesible y operativa para los usuarios y aplicaciones cuando se la necesita.

Aspectos clave para garantizarla:

- Tiempo de actividad (Uptime): Minimizar las caídas del servidor.
- Tolerancia a fallos: Usar técnicas como la replicación (ej: en MySQL, un servidor maestro se copia a uno o varios esclavos) para que, si uno falla, otro pueda tomar el control.
- Recuperación ante desastres: Realizar backups automáticos regulares (ej: con mysqldump) para restaurar los datos en caso de pérdida.
- Escalabilidad y Balance de Carga: Usar clusters (ej: MySQL InnoDB Cluster) para distribuir la carga y evitar la saturación.
- Monitoreo: Usar herramientas para detectar y alertar sobre problemas de rendimiento o recursos antes de que afecten la disponibilidad.

Parte 2: Transacciones y Propiedades ACID

1. ¿Qué es una transacción de base de datos? Enumere y describa sus características clave.

Una transacción es una secuencia de una o más operaciones SQL (como INSERT, UPDATE, DELETE) que se ejecutan como una única unidad lógica de trabajo.

Características clave:

- Unidad Indivisible: Todas las operaciones se ejecutan o ninguna lo hace.
- Consistencia: Lleva a la base de datos de un estado válido a otro.
- Aislamiento: Las operaciones de una transacción son temporales y no visibles para otras hasta que se confirman.
- Durabilidad: Una vez confirmada, los cambios son permanentes.

2. Para cada una de las propiedades ACID: defina, explique cómo se logra y proporcione un ejemplo de su violación.

- Atomicidad:
    - Definición: Garantiza que una transacción se trata como una unidad indivisible. O se ejecuta completamente o no se ejecuta en absoluto.
    - ¿Cómo se logra? El sistema usa un log de deshacer (undo log). Si la transacción falla, se usa este log para revertir todas las operaciones realizadas hasta ese punto.
    - Ejemplo de violación: En una transferencia, si se resta el dinero de la cuenta origen pero falla el sumarlo a la cuenta destino, y el sistema no revierte la

primera operación, el dinero simplemente desaparecería (violando la atomicidad).

- Consistencia:
    - Definición: Asegura que una transacción lleve la base de datos de un estado válido (que cumple todas las reglas) a otro estado válido.
    - ¿Cómo se logra? Mediante las restricciones de la base de datos (claves primarias, foráneas, CHECK) y la lógica de la aplicación. Si una transacción viola una restricción, se aborta.
    - Ejemplo de violación: Una transacción que intenta insertar un pedido con un id_cliente que no existe en la tabla Clientes. Si se permitiera, violaría la integridad referencial.

- Aislamiento:
    - Definición: Garantiza que las transacciones concurrentes (que se ejecutan al mismo tiempo) no interfieran entre sí. Cada una debe parecer que se ejecuta en aislamiento.
    - ¿Cómo se logra? Usando mecanismos como bloqueos (locking) o Control de Concurrencia Multiversión (MVCC).
    - Ejemplo de violación (Lectura Sucia): La Transacción A actualiza un saldo pero no hace COMMIT. La Transacción B lee ese saldo no confirmado. Si luego la Transacción A hace ROLLBACK, la Transacción B habrá tomado una decisión basada en un dato que nunca existió.

- Durabilidad:
    - Definición: Garantiza que una vez que una transacción se confirma (COMMIT), sus cambios son permanentes y sobreviven a cualquier fallo del sistema (corte de luz, caída del servidor).
    - ¿Cómo se logra? Usando un log de rehacer (redo log). Antes de confirmar, los cambios se escriben en este log en un almacenamiento no volátil (disco). Tras un fallo, se pueden reaplicar los cambios desde el log.
    - Ejemplo de violación: Un usuario recibe un mensaje de "Transferencia exitosa" (COMMIT), pero tras un corte de energía, el dinero vuelve a su cuenta origen porque los cambios no se persistieron correctamente en el disco.

3. Explique la importancia de las sentencias BEGIN, COMMIT y ROLLBACK en el contexto de la atomicidad.

Estas sentencias son fundamentales para controlar manualmente la atomicidad de una transacción.

- BEGIN: Marca el inicio de la unidad lógica de trabajo. A partir de aquí, todas las operaciones SQL forman parte de la misma transacción.

- COMMIT: Confirma toda la transacción. Si se ejecuta, significa que todas las operaciones desde el BEGIN se han realizado con éxito y sus cambios se hacen permanentes. Es el "éxito" de la atomicidad.

- ROLLBACK: Deshace toda la transacción. Si ocurre un error en cualquier operación dentro de la transacción, se ejecuta esta sentencia para revertir todos los cambios realizados desde el BEGIN, dejando la base de datos como si la transacción nunca hubiera empezado. Es el "fracaso" controlado de la atomicidad.

Sin ellas, cada sentencia SQL se confirmaría automáticamente, haciendo imposible revertir operaciones fallidas y violando la atomicidad.

Parte 3: Control de Concurrencia

1. ¿Qué es la concurrencia en bases de datos y cuáles son los desafíos principales que presenta en entornos multiusuario?

La concurrencia es la capacidad de que múltiples transacciones se ejecuten al mismo tiempo en un sistema de base de datos.

Desafíos principales:

- Mantener la integridad y consistencia de los datos cuando varias transacciones intentan leer y modificar la misma información simultáneamente.

- Evitar problemas como lecturas sucias, no repetibles, fantasmas y actualizaciones perdidas.

- Equilibrar el aislamiento con el rendimiento: Un aislamiento muy estricto (ej: SERIALIZABLE) garantiza consistencia pero reduce la concurrencia y el rendimiento.

2. Enumere y describa los cuatro problemas de concurrencia comunes

- Lectura Sucia: Ocurre cuando una transacción lee datos que han sido modificados por otra transacción que aún no se ha confirmado. Si esta última se revierte, la primera habrá leído datos que nunca fueron válidos.

- Lectura No Repetible: Ocurre cuando una transacción lee el mismo dato dos veces y obtiene valores diferentes porque otra transacción lo modificó y confirmó entre las dos lecturas.

- Lectura Fantasma: Ocurre cuando una transacción ejecuta la misma consulta dos veces y obtiene un conjunto de filas diferente porque otra transacción insertó o eliminó filas que cumplen la condición de la consulta.

- Actualización Perdida: Ocurre cuando dos transacciones leen el mismo dato y lo modifican basándose en el valor leído. La transacción que confirma en último lugar sobrescribe los cambios de la primera, "perdiendo" su actualización.

3. Niveles de aislamiento:

- **Niveles de aislamiento:**

Nivel	Lectura sucia	Lectura no repetible	Lectura fantasma
READ UNCOMMITTED	Permite	Permite	Permite
READ COMMITTED	Previene	Permite	Permite
REPEATABLE READ	Previene	Previene	Permite
SERIALIZABLE	Previene	Previene	Previene

- **Cómo se relaciona el nivel de aislamiento con el rendimiento de la base de datos?**

Existe una relación inversa. A mayor nivel de aislamiento (más protección contra anomalías), menor es la concurrencia y el rendimiento. Niveles como SERIALIZABLE requieren más bloqueos, haciendo que las transacciones esperen más entre sí. Niveles como READ UNCOMMITTED ofrecen máximo rendimiento pero a costa de la consistencia de los datos.

- **Demuestre cómo se establece un nivel de aislamiento para una transacción en SQL**

```sql
-- Establece el nivel de aislamiento para la sesion actual
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;

-- Inicia una transaccion
START TRANSACTION;

-- Operaciones SQL
SELECT *
FROM cuentas
WHERE id = 1;

UPDATE productos
SET stock = stock - 1
WHERE id = 10;

-- Confirmar la transaccion
COMMIT;
```

4. Describa los tres mecanismos de control de concurrencia. Explique los tipos de bloqueos.

1) Bloqueo (Locking): Restringe el acceso a un dato. Una transacción debe adquirir un bloqueo sobre un dato para leerlo o modificarlo.

    - Tipos:
        - Bloqueo Compartido (S): Para operaciones de lectura. Varias transacciones pueden tener un bloqueo compartido sobre el mismo dato al mismo tiempo.
        - Bloqueo Exclusivo (X): Para operaciones de escritura. Solo una transacción puede tener un bloqueo exclusivo sobre un dato. Impide que otras transacciones lean o escriban ese dato.

    - Cómo previene actualizaciones perdidas: En el ejemplo del Usuario A y B, cuando A comienza a actualizar un producto, adquiere un bloqueo exclusivo sobre esa fila. Si B intenta actualizar la misma fila, su transacción se bloquea y espera hasta que A libere el bloqueo (con COMMIT o ROLLBACK). Así, B siempre trabajará sobre la versión más reciente del dato, evitando sobrescribir los cambios de A.

2) Ordenamiento por Marca de Tiempo (Timestamp Ordering): A cada transacción se le asigna una marca de tiempo única. Las operaciones se ejecutan en un orden equivalente al de sus marcas de tiempo. Si una operación llega "tarde" (su marca de tiempo es anterior a la de una operación ya confirmada), se rechaza y se aborta la transacción.

3) Control de Concurrencia de Múltiples Versiones (MVCC): En lugar de bloquear los datos, el sistema mantiene múltiples versiones de un mismo dato. Una transacción de lectura ve una "instantánea" de los datos tal como estaban en el momento en que comenzó, incluso si otros los han modificado después. Esto permite lecturas sin bloqueos, mejorando mucho el rendimiento en entornos con muchas lecturas. MySQL (InnoDB) y PostgreSQL usan MVCC.

5. Qué es un bloqueo mutuo (deadlock) y cómo se resuelven?

Un deadlock es un punto muerto que ocurre cuando dos o más transacciones se bloquean mutuamente, esperando cada una a que la otra libere un recurso (ej: una fila de una tabla) que necesita para continuar. Ninguna puede avanzar.

- Ejemplo: La Transacción A tiene un bloqueo sobre la Fila 1 y necesita la Fila 2. La Transacción B tiene un bloqueo sobre la Fila 2 y necesita la Fila 1. Ambas esperan indefinidamente.

Cómo se resuelven?

Los sistemas de bases de datos los detectan automáticamente mediante algoritmos que buscan ciclos de espera. Cuando se detecta un deadlock, el sistema elige una de las transacciones como "víctima", la aborta (hace ROLLBACK) y libera todos sus bloqueos.

Esto permite que la otra transacción pueda continuar. La transacción abortada typically debe ser reintentada por la aplicación.

Parte 4: Aplicaciones en el Mundo Real y Preguntas para Investigar

1. Identifique y explique tres aplicaciones del mundo real donde las transacciones son críticas

- Sistemas Bancarios (Transferencias): Una transferencia implica debitar de una cuenta y acreditar en otra. Esta operación debe ser atómica (o se hacen ambos movimientos o ninguno) para evitar que el dinero se pierda o se cree de la nada. También debe ser duradera, una vez confirmada, no puede revertirse por un fallo.

- Comercio Electrónico (Procesamiento de Pedidos): Cuando un cliente finaliza una compra, se ejecuta una transacción que actualiza el inventario (restando stock), registra el pedido y procesa el pago. Si el pago falla, toda la transacción debe revertirse (ROLLBACK) para que el stock no disminuya y el pedido no quede registrado inconsistentemente.

- Sistemas de Reservas (Aerolíneas/Hoteles): Al reservar un asiento o habitación, la transacción debe verificar disponibilidad y bloquear el recurso. El aislamiento es crucial aquí para evitar la "doble venta" (que dos usuarios reserven el mismo recurso al mismo tiempo), lo que se conoce como una actualización perdida.

2. Preguntas de Investigación

Pregunta 1: Compare el control de concurrencia optimista vs. pesimista

El control de concurrencia pesimista asume que los conflictos entre transacciones son probables, por lo que bloquea los recursos de antemano para prevenirlos. Es como decir: "Voy a cambiar este dato, así que lo bloqueo para que nadie más lo toque hasta que yo termine". Se basa en el mecanismo de bloqueos (locking). Es ideal para entornos con alta contienda de datos, donde es frecuente que varias transacciones intenten modificar los mismos registros.

El control de concurrencia optimista asume que los conflictos son raros. En lugar de bloquear, permite que las transacciones accedan a los datos libremente. Solo al final, cuando la transacción intenta confirmar (COMMIT), verifica si hubo un conflicto (ej: comparando versiones de los datos). Si no lo hubo, se confirma. Si hubo un conflicto, se aborta y se debe reintentar. Se basa a menudo en MVCC y marcas de tiempo. Es ideal para entornos con muchas lecturas y pocas escrituras, donde los bloqueos ralentizarían innecesariamente el sistema.

¿Cuándo usar uno sobre el otro?

- Usar Pesimista: Cuando la tasa de conflictos es alta y el costo de abortar y reintentar una transacción (en el modelo optimista) sería mayor que el de gestionar bloqueos. Ej: sistema de subastas donde un ítem tiene muchos postores al mismo tiempo.

- Usar Optimista: Cuando la tasa de conflictos es baja. Ofrece mucho mejor rendimiento porque evita la sobrecarga de los bloqueos. Ej: un blog donde los usuarios comentan mucho pero editan poco sus comentarios.

Fuente:

- https://learn.microsoft.com/es-es/sql/relational-databases/sql-server-transaction-locking-and-row-versioning-guide?view=sql-server-ver16

Pregunta 2: Investigue cómo un DBMS específico (PostgreSQL) implementa ACID y sus niveles de aislamiento predeterminados

- DBMS Elegido: PostgreSQL
- Implementación de ACID en PostgreSQL:
    - Atomicidad y Durabilidad: Se logran mediante el Write-Ahead Logging (WAL). Todas las modificaciones se escriben primero en el WAL (en disco) antes de aplicarse a las tablas principales. Esto permite recuperar transacciones confirmadas tras un fallo (Durabilidad) y también deshacer transacciones no confirmadas (Atomicidad).
    - Consistencia: PostgreSQL hace cumplir fuertemente la consistencia mediante restricciones (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL). Una transacción que viole cualquier restricción será abortada.
    - Aislamiento: PostgreSQL utiliza MVCC (Multi-Version Concurrency Control) de forma nativa para lograr el aislamiento. Cada transacción ve una "instantánea" de los datos consistente en un punto en el tiempo, lo que permite lecturas sin bloqueos.
- Niveles de Aislamiento Predeterminados:
    - El nivel de aislamiento predeterminado en PostgreSQL es READ COMMITTED.
    - Esto difiere de MySQL (InnoDB), que por defecto usa REPEATABLE READ.
- Diferencias significativas con la descripción general del módulo:
    - Nivel por Defecto: Como se mencionó, PostgreSQL usa READ COMMITTED por defecto, que es menos restrictivo que el REPEATABLE READ de MySQL. Esto refleca un enfoque diferente en el equilibrio entre consistencia y rendimiento.

- Prevención de Lecturas Fantasma: En el nivel REPEATABLE READ de PostgreSQL, además de prevenir lecturas sucias y no repetibles, también previene las lecturas fantasma para la mayoría de las consultas. Esto es una ventaja sobre el estándar SQL, que permite fantasmas en este nivel. Lo logra tomando una instantánea de los datos al inicio de la transacción y manteniéndola.

- MVCC Nativo vs. Implementación: Mientras que MySQL (InnoDB) también usa MVCC, la implementación en PostgreSQL es más "pura" y está profundamente integrada en el núcleo del motor, lo que se traduce en un comportamiento muy robusto y predecible en escenarios complejos de concurrencia.

Fuentes:

- https://www.postgresql.org/docs/current/transaction-iso.html (traducida)