Unidad 7 – Trabajo Práctico: Integridad, Consistencia, Disponibilidad, Transacciones y Control de Concurrencia

Morales Ramiro

Parte 1: Fundamentos de la Integridad y Consistencia

1. Integridad de los datos

La integridad de los datos es la garantía de que la información almacenada en una base de datos es correcta, coherente y confiable durante todo su ciclo de vida. Es crucial porque evita errores, duplicaciones o datos contradictorios que podrían afectar las operaciones o decisiones de una organización.

Tipos de restricciones de integridad:

- Integridad de entidad: asegura que cada registro tenga una identidad única.
  Ejemplo: en una tabla de clientes, el campo id_cliente no puede repetirse ni ser nulo.

- Integridad referencial: garantiza que las relaciones entre tablas sean válidas.
  Ejemplo: si una tabla de pedidos tiene un campo id_cliente, ese valor debe existir en la tabla de clientes.

- Integridad de dominio: controla que los valores ingresados estén dentro de un conjunto permitido.
  Ejemplo: el campo “edad” solo puede aceptar valores numéricos entre 0 y 120.

2. Consistencia

La consistencia asegura que la base de datos pase de un estado válido a otro también válido después de ejecutar una operación o transacción.
En el ejemplo de una transferencia bancaria, cuando un cliente transfiere $100 a otro, la base de datos debe restar $100 de una cuenta y sumarlos en la otra. Si la operación se interrumpe, no puede quedar un estado donde el dinero desaparece o se duplica.

La consistencia se mantiene gracias a las reglas de integridad, validaciones y restricciones que garantizan que los datos siempre cumplan las condiciones definidas.

3. Disponibilidad

La disponibilidad significa que la base de datos está accesible y operativa para los usuarios cuando la necesitan.
Para lograrlo se aplican prácticas como:

- Redundancia de servidores (replicación) para que el sistema siga funcionando si uno falla.
- Copias de seguridad periódicas.
- Mantenimiento y monitoreo continuo para detectar fallas.
- Sistemas distribuidos o clústeres, que permiten atender múltiples solicitudes sin interrupciones.

Transacciones y Propiedades ACID

1. Transacción de base de datos

Una transacción es un conjunto de operaciones que se ejecutan como una unidad unica..
Su objetivo es garantizar que todas las operaciones se completen correctamente o, en caso de error, que ninguna tenga efecto.
Sus características clave son:

- Atomicidad: todo o nada.
- Consistencia: la base de datos pasa de un estado válido a otro.
- Aislamiento: las transacciones no interfieren entre sí.
- Durabilidad: una vez confirmada, la transacción persiste aunque haya fallos.

2. Propiedades ACID

a) Atomicidad

- Definición: asegura que todas las operaciones dentro de una transacción se ejecuten completamente o se deshagan.
- Cómo se logra: mediante mecanismos como COMMIT y ROLLBACK.
- Ejemplo: si se actualizan dos cuentas bancarias y una falla, ninguna de las dos debe quedar modificada.

b) Consistencia

- Definición: garantiza que la base de datos mantenga sus reglas de integridad antes y después de la transacción.
- Cómo se logra: validando reglas, restricciones y claves foráneas.
- Ejemplo: no se puede registrar una venta con un producto que no existe.

c) Aislamiento

- Definición: impide que los cambios de una transacción sean visibles por otras hasta que se confirme.
- Cómo se logra: con bloqueos o niveles de aislamiento.
- Ejemplo: dos usuarios no deberían poder modificar al mismo tiempo el saldo de una misma cuenta sin control.

d) Durabilidad

- Definición: una vez confirmada una transacción, sus efectos son permanentes.
- Cómo se logra: guardando los cambios en disco o mediante logs de recuperación.
- Ejemplo: aunque se apague el servidor, los cambios confirmados siguen existiendo.

3. BEGIN, COMMIT y ROLLBACK

Estas sentencias controlan la atomicidad de una transacción:

- BEGIN: marca el inicio de la transacción.
- COMMIT: confirma los cambios si todo se ejecutó correctamente.

- ROLLBACK: revierte los cambios si ocurre un error.

Parte 3: Control de Concurrencia

1. Concurrencia

La concurrencia ocurre cuando varios usuarios acceden o modifican la base de datos al mismo tiempo.
El desafío principal es evitar inconsistencias, bloqueos o pérdidas de información por operaciones simultáneas.

2. Problemas comunes de concurrencia

- Lecturas sucias: una transacción lee datos que aún no fueron confirmados por otra.
- Lecturas no repetibles: una transacción lee el mismo dato dos veces y obtiene resultados distintos porque otra lo modificó en el medio.
- Lecturas fantasma: aparecen o desaparecen filas entre dos consultas debido a inserciones o borrados de otras transacciones.
- Actualizaciones perdidas: dos transacciones modifican el mismo registro y una sobrescribe los cambios de la otra.

3. Niveles de aislamiento

Los principales niveles de aislamiento son:

1. Read Uncommitted: permite leer datos sin confirmar (riesgo alto de lecturas sucias).
2. Read Committed: solo permite leer datos confirmados (evita lecturas sucias).
3. Repeatable Read: evita lecturas no repetibles, pero puede haber lecturas fantasma.
4. Serializable: el nivel más alto, evita todos los problemas de concurrencia.

Relación con el rendimiento:
A mayor aislamiento, mayor seguridad pero menor rendimiento, porque se necesitan más bloqueos y control.

4. Mecanismos de control de concurrencia

- Bloqueo (Locking): impide que varias transacciones modifiquen el mismo dato simultáneamente.
    - *Bloqueo compartido (S)*: permite lectura múltiple.
    - *Bloqueo exclusivo (X)*: solo una transacción puede escribir.
      Ejemplo: si el usuario A edita un registro, el usuario B debe esperar a que A confirme o cancele para evitar una actualización perdida.
- Ordenamiento por marca de tiempo (Timestamp): cada transacción recibe una marca temporal. Las operaciones se ejecutan según ese orden para mantener coherencia.
- Control de concurrencia multiversión (MVCC): crea versiones de los registros para que las lecturas no bloqueen escrituras. Así, cada usuario ve un “estado consistente” de los datos.

---

5. Bloqueo mutuo

Un *deadlock* ocurre cuando dos transacciones esperan recursos que la otra posee, generando un ciclo donde ninguna puede continuar.
Se resuelve mediante:

- Detección automática: el sistema identifica el bloqueo y cancela una de las transacciones.
- Prevención: ordenando la adquisición de recursos o limitando el tiempo de espera.

Parte 4: Aplicaciones y Preguntas de Investigación

1. Aplicaciones del mundo real

1. Bancos y sistemas financieros: las transacciones aseguran que los movimientos de dinero sean consistentes y no se dupliquen.
2. Comercio electrónico: al realizar una compra, se deben actualizar existencias, pagos y pedidos sin errores.

3. Sistemas hospitalarios: al registrar diagnósticos o medicaciones, los datos deben ser exactos y coherentes en todo momento.

2. Preguntas de investigación

a) Estrategias de concurrencia: optimista vs pesimista

- Control pesimista: bloquea los datos antes de modificarlos para evitar conflictos. Es útil en entornos con muchas colisiones, como sistemas bancarios.

- Control optimista: asume que los conflictos son raros y solo valida al final si los datos fueron modificados por otros. Es mejor para sistemas con muchas lecturas y pocas escrituras, como aplicaciones de consulta.

b) Implementación de ACID en PostgreSQL
PostgreSQL cumple con las propiedades ACID mediante su motor transaccional y el uso de MVCC (control multiversión).

- Atomicidad: se maneja con COMMIT y ROLLBACK.

- Consistencia: asegura que las reglas de integridad se cumplan siempre.

- Aislamiento: por defecto usa el nivel *Read Committed*, pero permite configurarlo hasta *Serializable*.

- Durabilidad: garantiza que los cambios confirmados se guarden en el *Write-Ahead Log (WAL)*, que permite recuperar el estado tras un fallo.