**Integridad y Consistencia de Datos**

**Consistencia en Bases de Datos**

**Integridad**: garantiza que los datos sean precisos, confiables y uniformes en todo momento y en todos los sistemas.

refiere al conjunto de procesos, reglas y normas que tienen como fin mantener los datos completos, precisos, coherentes y seguros a lo largo de su ciclo de vida.

**Consistencia:** Duración, estabilidad, solidez.

**Factores que afectan la integridad y consistencia en una base de datos:**

• Las amenazas **físicas** abarcan todos aquellos eventos que pueden causar daño directo al hardware o a los medios de almacenamiento donde residen los datos

•Las amenazas **lógicas** son aquellas que surgen de errores en el software, procesos o acciones humanas que pueden comprometer la integridad y consistencia de los datos. Actualizaciones incompletas o transacciones interrumpidas, pueden dejar los datos en un estado inconsistente

**Integridad en Bases de Datos:**

La integridad de datos hace referencia al conjunto de procesos, reglas y normas que tienen como fin mantener los datos completos, precisos, coherentes y seguros a lo largo de su ciclo de vida.

● Integridad Física: Es la protección de los datos en su formato físico.

● Integridad Lógica: También se enfoca en la protección de los datos, pero desde el punto de vista lógico

**✅ 1. Integridad de dominio**

Asegura que los **valores de una columna sean válidos**, es decir, del tipo y formato correcto (por ejemplo, que la edad sea un número o el email tenga formato de texto).  
👉 *Se logra usando tipos de datos, restricciones como NOT NULL, UNIQUE, etc.*

**✅ 2. Integridad de identidad**

Garantiza que **cada registro tenga un identificador único**, evitando duplicados.  
👉 *Se implementa usando claves primarias (PRIMARY KEY) y autoincrementos (AUTO\_INCREMENT).*

**✅ 3. Integridad referencial**

Se asegura de que **las relaciones entre tablas sean coherentes**.  
👉 *Por ejemplo, que ciudad\_id en la tabla clientes exista realmente en la tabla ciudades, usando FOREIGN KEY.*

**✅ 4. Integridad definida por el usuario**

Son **reglas personalizadas** que los usuarios definen según las necesidades del negocio.  
👉 *Por ejemplo, que todos los clientes sean mayores de edad: CHECK (edad >= 18).*

**Aplicación de integridad - Restricciones en tablas**

CONSTRAINT -> RESTRICCIONES

**● NOT NULL:** Garantizando que un campo específico no posea valores NULL.

● **UNIQUE:** Garantiza que cada valor es único en la columna; es decir, no puede haber dos registros con dos valores iguales en la columna.

● **PRIMARY KEY**: Garantiza que cada registro en una tabla tenga un identificador único. Las claves primarias deben contener valores UNIQUE y no pueden contener valores NULL. Una tabla puede tener solo UNA clave primaria; y en la tabla, esta clave primaria puede consiste en columnas únicas o múltiples (campos).

● **FOREIGN KEY:** Garantiza la correcta asociación de una tabla con otra, evitando acciones que destruyan los enlaces entre tablas.

● **CHECK:** Permite establecer condiciones específicas sobre los valores de un campo.

● **DEFAULT**: Si no se especifica el valor del campo al momento de agregar un nuevo registro, entonces se asignará el valor indicado en la cláusula

**Aplicación de integridad – Trigger**

Un Trigger (disparador) es un objeto de la Base de Datos que está asociado con una tabla, y que se activa cuando se produce un evento particular en ella. En definitiva, Un Trigger es una acción en cadena que empieza cuando un evento específico ocurre sobre una tabla específica Los eventos disparadores pueden ser tres:

● INSERT

● DELETE

● UPDATE

Un Trigger puede activarse:

• automáticamente antes (BEFORE)

• después (AFTER) de que ocurra el evento.

Tipos de Triggers:

● Disparadores de fila (Row Triggers): Se ejecutan una vez para cada fila

● Disparadores de Sentencia (Statement Triggers): Se ejecuta solo una vez(no disponibleMySQ)

**Limitaciones en la sentencia de un Trigger**

\* El disparador no puede referirse a tablas directamente por su nombre , se pueden emplear las palabras clave OLD y NEW.

● OLD se refiere a un registro existente que va a borrarse o que va a actualizarse antes de que esto ocurra.

● NEW se refiere a un registro nuevo que se insertará o a un registro modificado luego de que ocurre la modificación

\* El disparador no puede invocar procedimientos almacenados utilizando la sentencia CALL.

\* El disparador no puede utilizar sentencias que inicien o finalicen una transacción, tal como START TRANSACTION, COMMIT, o ROLLBACK

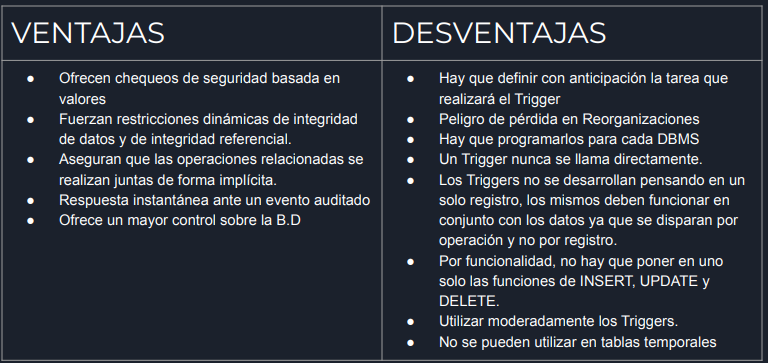
**Estructura Básica de un Trigger**

**Llamada de activación:** es la sentencia que permite “disparar” el código a ejecutar.

**Restricción:** es la condición necesaria para ejecutar el código. Esta restricción puede ser de tipo condicional o de tipo nulidad.

**Acción a ejecutar:** es la sentencia de instrucciones a ejecutar una vez que se han cumplido las condiciones iniciales.

CREATE TRIGGER [NOMBRE\_TRIGGER] [AFTER|BEFORE] [INSERT|UPDATE|DELETE] ON [NOMBRE\_TABLA] FOR EACH ROW [SENTENCIA]

****

**Procedimientos Almacenados y Funciones**

Un **procedimiento almacenado** es un conjunto de instrucciones SQL que se almacenan en el servidor de base de datos y pueden ser invocados mediante un nombre específico.

Estos procedimientos pueden recibir parámetros, ejecutar operaciones complejas y devolver múltiples resultados.

Ventajas de los Procedimientos Almacenados

• Rendimiento mejorado: Las consultas precompiladas reducen el tiempo de ejecución.

• Seguridad: Permiten controlar el acceso a las tablas subyacentes.

• Reutilización de código: Centraliza la lógica de negocio en la base de datos.

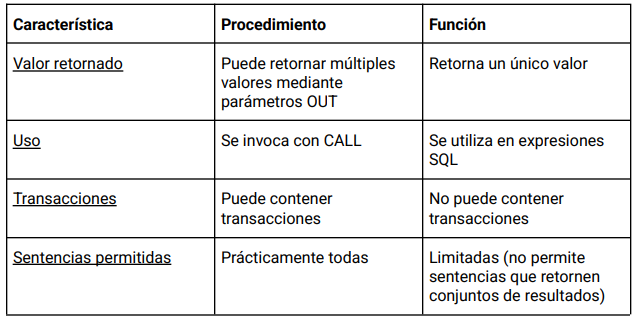
• Reducción del tráfico de red: Solo se envía la llamada al procedimiento, no todas las instrucciones SQL.

• Mantenimiento: Facilita la modificación de lógica sin cambiar el código de aplicación.

• Transacciones: Simplifican la gestión de transacciones complejas.

**Función Almacenada:** Una función almacenada es similar a un procedimiento, pero está diseñada para calcular y devolver un único valor. Se utilizan típicamente en expresiones SQL.

**Diferencias entre Procedimientos y Funciones**



**SEGURIDAD**

• Validar siempre los parámetros de entrada para prevenir inyección SQL.

• Utilizar prepared statements cuando se construyan consultas dinámicas.

• Implementar el principio de privilegio mínimo en los permisos.

• Encriptar datos sensibles antes de almacenarlos.

**Transacción**

es una secuencia de operaciones que se ejecutan como una unidad lógica de trabajo indivisible. Podemos entenderla como un "paquete" de operaciones que deben completarse en su totalidad o no ejecutarse en absoluto.

**Propiedades ACID**

**Atomicidad (Atomicity):** garantiza que todas las operaciones dentro de una transacción se traten como una unidad indivisible

**Consistencia (Consistency**): segura que una transacción sólo puede llevar la base de datos de un estado válido a otro estado igualmente válido, respetando todas las reglas, restricciones e integridad referencial definidas

**Aislamiento (Isolation):** El aislamiento determina cómo se manejan las transacciones concurrentes y cómo los cambios realizados por una transacción son visibles para otras transacciones ejecutándose simultáneamente.

- *READ UNCOMMITTED(MYSQL POR DEFECTO)*: Permite leer datos no confirmados por otras transacciones, lo que puede causar lecturas sucias

- *READ COMMITTED:* Solo permite leer datos confirmados, evitando las lecturas sucias. Sin embargo, puede producir lecturas no repetibles

- *REPEATABLE READ:* Garantiza que si una transacción lee un dato, seguirá viendo el mismo valor aunque otras transacciones lo modifiquen y confirmen. Previene lecturas no repetibles, pero puede provocar lecturas fantasma

- *SERIALIZABLE*: El nivel más estricto, hace que las transacciones se comporten como si se ejecutaran en serie, una después de otra, eliminando todos los problemas de concurrencia posibles, pero reduciendo significativamente el rendimiento

**Durabilidad (Durability:** garantiza que una vez que una transacción ha sido confirmada, sus cambios permanecen en la base de datos incluso ante fallos del sistema, como cortes de energía o errores de hardware.

**Control de Concurrencia y Bloqueos**

El control de concurrencia es el mecanismo que permite que múltiples usuarios accedan y modifiquen los datos simultáneamente sin crear inconsistencias.

**Tipos de Bloqueos en MySQL**

1. **Bloqueos compartidos (Shared locks):** Permiten a múltiples transacciones leer los mismos datos simultáneamente, pero impiden que escriban sobre ellos.
2. **Bloqueos exclusivos (Exclusive locks):** Impiden que otras transacciones lean o escriban los datos bloqueados.
3. **Bloqueos de intención (Intent locks):** Indican que una transacción planea adquirir un bloqueo más específico sobre ciertos datos.
4. **Bloqueos a nivel de fila (Row-level locks):** Permiten bloquear filas individuales en lugar de tablas completas, mejorando el rendimiento en entornos con alta concurrencia.

**Control de Concurrencia Multiversión (MVCC):**

permite a múltiples transacciones trabajar con los mismos datos sin interferirse entre sí. MVCC funciona manteniendo diferentes versiones de los datos. Cuando una transacción inicia, captura una instantánea del estado actual de la base de datos y trabaja con esa versión. Esto permite que otras transacciones sigan operando con su propia versión de los datos, evitando conflictos.

**Para optimizar la concurrencia, es recomendable:**

● Mantener las transacciones lo más cortas posible

● Evitar transacciones de larga duración que mantengan bloqueos por períodos extensos

● Acceder a las tablas en el mismo orden en todas las transacciones para evitar interbloqueos

**MySQL ofrece varias sentencias para trabajar con transacciones:**

1. START TRANSACTION / BEGIN: Inicia una nueva transacción.

2. COMMIT: Confirma todos los cambios realizados durante la transacción.

3. ROLLBACK: Deshace todos los cambios realizados durante la transacción.

4. SET autocommit: Controla si las operaciones se confirman automáticamenteo requieren un COMMIT explícito.

**Uso de Manejadores de Excepciones**

• Puntos de Guardado (Savepoints) Los puntos de guardado permiten definir marcadores dentro de una transacción para poder realizar rollbacks parciales:

**Problemas de Concurrencia**

Cuando múltiples transacciones operan simultáneamente, pueden surgir varios problemas:

1. Lecturas sucias (Dirty reads): Una transacción lee datos modificados por otra transacción que aún no ha sido confirmada.

2. Lecturas no repetibles (Non-repeatable reads): Una transacción lee el mismo registro dos veces y obtiene valores diferentes porque otra transacción ha modificado los datos entre las lecturas.

3. Lecturas fantasma (Phantom reads): Una transacción ejecuta la misma consulta dos veces y la segunda vez aparecen filas adicionales que cumplen los criterios de búsqueda porque otra transacción ha insertado nuevos registros.

4. Actualizaciones perdidas (Lost updates): Dos transacciones leen el mismo registro, lo modifican y lo escriben, causando que la última actualización sobrescriba la primera.

5. Interbloqueo (Deadlock): Dos transacciones se bloquean mutuamente porque cada una espera recursos que la otra tiene bloqueados. MySQL detecta automáticamente los interbloqueos y resuelve la situación abortando una de las transacciones.

**Estrategias para Evitar Problemas de Concurrencia**

1. Seleccionar el nivel de aislamiento adecuado según los requisitos de la aplicación:

2. Optimizar la estructura de las transacciones:

3. Utilizar bloqueos explícitos cuando sea necesario.

4. Acceder a las tablas en un orden consistente en todas las transacciones para evitar interbloqueos.

**Optimizar la estructura de las transacciones:**

Mantener las transacciones lo más cortas posible

Minimizar el número de operaciones dentro de una transacción

Evitar operaciones de usuario (como esperar entrada) dentro de una transacción

**Índices y Optimización de Consultas en Bases de Datos**