

## 📘 Integridad, Consistencia y Disponibilidad en Bases de Datos (MySQL)

### 📘 Introducción

En MySQL, la **integridad de los datos**, la **consistencia** y la **disponibilidad** son fundamentales para garantizar que las operaciones realizadas sobre la base de datos produzcan resultados válidos, confiables y accesibles.

### 🔍 1. Integridad de los datos en MySQL

La integridad asegura que los datos sean válidos y coherentes, y que cumplan con las reglas definidas en el esquema de la base.

#### ☑ Tipos de integridad:

##### A. Integridad de entidad

Cada fila debe poder identificarse de forma única, mediante una **clave primaria**.

🔧 En MySQL:

```
CREATE TABLE Cliente (
```

```
    id INT PRIMARY KEY AUTO_INCREMENT,  
    nombre VARCHAR(100) NOT NULL  
,
```

##### B. Integridad referencial

Impide que existan referencias a registros que no existen en otra tabla. Se logra mediante **claves foráneas** (foreign keys).

🔧 En MySQL:

```
CREATE TABLE Pedido (
```

```
    id INT PRIMARY KEY AUTO_INCREMENT,  
    cliente_id INT,  
    FOREIGN KEY (cliente_id) REFERENCES Cliente(id)  
        ON DELETE CASCADE  
        ON UPDATE CASCADE  
,
```



## C. Integridad de dominio

Controla los valores permitidos para una columna, a través del tipo de dato, restricciones NOT NULL, CHECK, etc.

*Ejemplo en MySQL:*

```
CREATE TABLE Producto (
    id INT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    precio DECIMAL(10,2) CHECK (precio > 0)
);
```

*Nota:* MySQL evalúa CHECK pero en versiones anteriores a 8.0 no se aplicaba efectivamente.

## D. Integridad de negocio o de usuario

Define **reglas específicas del contexto de uso** que no siempre pueden imponerse a nivel de base de datos, pero sí pueden controlarse mediante procedimientos o lógica de aplicación.

*Ejemplo:* Un cliente no puede hacer más de 5 pedidos por día. Esta lógica podría controlarse desde un trigger o una capa intermedia.

## 2. Consistencia

La consistencia garantiza que la base de datos pase siempre de un **estado válido a otro estado válido**, cumpliendo las restricciones definidas.

- Si se viola alguna restricción (clave foránea, tipo de dato, dominio), la transacción falla y se revierte.
- En MySQL, esto se logra principalmente usando **InnoDB** junto con restricciones explícitas.

*Ejemplo típico en MySQL:*

Una transacción que debita dinero de una cuenta y lo acredita en otra debe:

- Respetar que el saldo no sea negativo (CHECK (saldo >= 0))
- Validar que ambas cuentas existan (clave foránea)

Si una operación falla, se hace ROLLBACK para evitar una base inconsistente.

## 3. Disponibilidad

La disponibilidad se refiere a que la base de datos esté **accesible y operativa** en todo momento para los usuarios y aplicaciones.

**Factores que afectan la disponibilidad:**

- Caídas del servidor



- Problemas de red
- Contención de recursos (por transacciones largas o mal gestionadas)

## 💡 Estrategias para mejorar la disponibilidad en MySQL:

### 1. Replicación:

- MySQL permite replicar datos de un servidor maestro a uno o varios esclavos.
- Se puede usar para alta disponibilidad o para balanceo de lectura.

### 2. Backups automáticos:

- Se recomienda realizar copias de seguridad completas y diferenciales.
- Herramientas: mysqldump, mysqlpump, MySQL Enterprise Backup.

### 3. Clusterización (MySQL Group Replication / InnoDB Cluster):

- Proporciona tolerancia a fallos y failover automático.

### 4. Monitoreo y alertas:

- Permiten anticipar fallos mediante herramientas como MySQL Enterprise Monitor, Percona Monitoring and Management, o soluciones en la nube.

## ⌚ Transacciones y Propiedades ACID en MySQL

### 🕒 ¿Qué es una transacción?

Una **transacción** en MySQL es una unidad lógica de trabajo que agrupa una o más operaciones SQL. Todas las operaciones de la transacción deben ejecutarse de forma **completa (COMMIT)** o no ejecutarse en absoluto (**ROLLBACK**), manteniendo la base de datos en un estado válido.

Las transacciones son soportadas por el motor **InnoDB**, que es el motor transaccional por defecto en MySQL.

### ❖ Propiedades ACID en MySQL

El modelo **ACID** define los principios que garantizan que las transacciones sean confiables:

#### ✓ 1. Atomicidad

Toda la transacción se ejecuta como una unidad indivisible. Si una parte falla, **todo se revierte**.

🔧 MySQL lo logra con el uso del log de deshacer (undo log).

🧠 Ejemplo: Si se intenta transferir dinero y la segunda actualización falla, la primera también se revierte automáticamente.

#### ✓ 2. Consistencia

La base de datos pasa de un estado válido a otro también válido.

- Se respetan restricciones de claves primarias, foráneas, checks, etc.



- Se garantiza que **no se violen reglas del modelo de datos**.

🔧 *En MySQL se logra a través del motor InnoDB y las restricciones definidas en el esquema.*

### 3. Aislamiento

Cada transacción actúa como si fuera la única en el sistema.

- Las operaciones de una transacción no son visibles para otras hasta que se hace **COMMIT**.
- Se evita que las transacciones interfieran entre sí.

🔧 *MySQL usa MVCC (Control de Concurrencia Multiversión) para lograrlo.*

#### Niveles de aislamiento en MySQL:

Nivel	¿Permite lectura sucia?	¿Lectura no repetible?	¿Fantasmas?
READ UNCOMMITTED	Sí	Sí	Sí
READ COMMITTED	No	Sí	Sí
REPEATABLE READ	No	No	Sí
SERIALIZABLE	No	No	No

🔧 *Por defecto, MySQL usa REPEATABLE READ (puede cambiar según configuración).*

### 4. Durabilidad

Una vez confirmado un cambio, **permanece en la base de datos**, incluso si hay una caída del sistema.

🔧 *InnoDB usa el Redo Log y escritura anticipada en disco para garantizarlo.*

### Sentencias SQL para transacciones en MySQL

START TRANSACTION;

-- operaciones SQL

UPDATE cuentas SET saldo = saldo - 100 WHERE id = 1;

UPDATE cuentas SET saldo = saldo + 100 WHERE id = 2;

COMMIT;

- START TRANSACTION; o BEGIN; → inicia la transacción
- COMMIT; → confirma los cambios
- ROLLBACK; → revierte todos los cambios si ocurre un error

### Estados posibles de una transacción

1. **Activa:** la transacción ha comenzado y está ejecutando instrucciones.
2. **Parcialmente completada:** todas las operaciones fueron ejecutadas, falta confirmar.



3. **Fallida:** ocurrió un error antes de finalizar.
4. **Abortada:** se hizo ROLLBACK.
5. **Confirmada:** se hizo COMMIT y los cambios fueron persistidos.

## Control de Conurrencia en Bases de Datos (MySQL)

### Introducción

En sistemas multiusuario como MySQL, varias transacciones pueden ejecutarse al mismo tiempo. Esto puede generar **conflictos** si se accede simultáneamente a los mismos datos. El **control de concurrencia** se encarga de evitar que estas situaciones afecten la **consistencia e integridad** de los datos.

### ¿Qué es la concurrencia?

La **concurrencia** es la capacidad de ejecutar múltiples transacciones simultáneamente. Aunque mejora el rendimiento, puede ocasionar errores si no se controla correctamente.

### Problemas comunes de concurrencia

#### 1. Lectura sucia (Dirty Read)

Una transacción **lee datos modificados por otra transacción que aún no se confirmó**.

 *Ejemplo:* Transacción A actualiza un precio, transacción B lo lee antes del COMMIT. Si A hace ROLLBACK, B leyó un valor que nunca existió oficialmente.

#### 2. Lectura no repetible (Non-repeatable Read)

Una transacción lee un mismo dato dos veces, y **obtiene distintos resultados** porque otra transacción lo modificó entre lecturas.

 *Ejemplo:* Transacción A consulta un saldo. Luego transacción B lo modifica y confirma. A vuelve a consultar y el saldo ha cambiado.

#### 3. Lectura fantasma (Phantom Read)

Una transacción ejecuta una consulta y luego, al repetirla, **aparecen nuevos registros** que antes no estaban, debido a otra transacción concurrente.

 *Ejemplo:* A consulta cuántos empleados ganan más de \$1000. B inserta uno nuevo en ese rango. A repite la consulta y obtiene un resultado distinto.

#### 4. Actualización perdida (Lost Update)

Dos transacciones **modifican el mismo dato**. Una sobrescribe los cambios de la otra sin saberlo.

 *Ejemplo:* Usuario A y B actualizan el mismo perfil al mismo tiempo. El último en confirmar sobrescribe los cambios del otro.

## Niveles de aislamiento en MySQL

MySQL permite configurar diferentes **niveles de aislamiento** que controlan qué fenómenos de concurrencia se pueden producir:

Nivel	Dirty Read	Non-repeatable Read	Phantom Read
READ UNCOMMITTED	<input checked="" type="checkbox"/> Sí	<input checked="" type="checkbox"/> Sí	<input checked="" type="checkbox"/> Sí
READ COMMITTED	<input checked="" type="checkbox"/> No	<input checked="" type="checkbox"/> Sí	<input checked="" type="checkbox"/> Sí
REPEATABLE READ (*)	<input checked="" type="checkbox"/> No	<input checked="" type="checkbox"/> No	<input checked="" type="checkbox"/> Sí
SERIALIZABLE	<input checked="" type="checkbox"/> No	<input checked="" type="checkbox"/> No	<input checked="" type="checkbox"/> No

(\*) REPEATABLE READ es el valor predeterminado en MySQL (InnoDB).

## Establecer el nivel de aislamiento en MySQL

```
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

```
START TRANSACTION;
```

```
-- operaciones SQL
```

```
COMMIT;
```

También se puede usar READ COMMITTED, SERIALIZABLE o READ UNCOMMITTED según necesidad.

## Mecanismos de control de concurrencia en MySQL

### A. Bloqueo (Locking)

MySQL/InnoDB utiliza **bloqueos a nivel de fila** por defecto, lo que permite mayor concurrencia.

**Tipos:**

- **Bloqueo compartido (S):** permite que otros lean pero no escriban.
- **Bloqueo exclusivo (X):** impide cualquier acceso de otras transacciones.

*Ejemplo práctico (modo explícito):*

```
SELECT * FROM productos WHERE id = 5 FOR UPDATE;
```

Esto bloquea la fila hasta que se haga COMMIT o ROLLBACK.

### B. Control por marcas de tiempo (Timestamp Ordering)

Aunque no es el método principal en MySQL, el orden lógico de ejecución puede lograrse por medio de control de versiones y marcas de tiempo internas para decidir si una transacción debe esperar, abortar o continuar.



### 💡 C. MVCC (Control de concurrencia multiversión)

MySQL (con InnoDB) **usa MVCC** para permitir que las transacciones lean una **versión consistente de los datos**, sin bloquear.

- Las transacciones **ven los datos como estaban al momento de comenzar**, incluso si otro usuario los modifica después.
- Mejora el rendimiento porque permite **lectura sin bloqueo**.

### ⌚ ¿Qué es un deadlock?

Un **deadlock** ocurre cuando **dos o más transacciones se bloquean mutuamente** esperando recursos que la otra no libera. Ninguna puede avanzar.

🧠 *Ejemplo:*

- A bloquea la fila 1 y espera la fila 2.
- B bloquea la fila 2 y espera la fila 1.

### 🛠 ¿Cómo lo maneja MySQL?

- MySQL detecta automáticamente los deadlocks.
- Cancela una de las transacciones y lanza un error: ERROR 1213 (40001): Deadlock found.

🔧 *Recomendación:* diseñar las transacciones para que accedan a los recursos siempre en el mismo orden y que sean lo más cortas posibles.

### 💡 Buenas prácticas para evitar deadlocks:

- Acceder a los recursos **en el mismo orden** en todas las transacciones.
- Mantener las transacciones **lo más cortas posible**.
- Usar **índices** para reducir el alcance de los bloqueos.

## 🌐 Aplicaciones Reales

### 📘 Introducción

Las bases de datos transaccionales como **MySQL con el motor InnoDB** están diseñadas para operar de forma segura y eficiente en entornos donde múltiples usuarios acceden simultáneamente. En estos entornos, es fundamental garantizar **la integridad de los datos, la ejecución confiable de transacciones y un buen manejo de la concurrencia**.

### ❖ 1. Aplicaciones reales donde las transacciones son esenciales

Las transacciones son fundamentales en escenarios donde se requiere **consistencia total y recuperación ante fallos**. Algunas aplicaciones comunes son:

## A. Sistemas bancarios

- Operaciones como transferencias de fondos, pagos y movimientos de cuenta se realizan mediante transacciones.
- Es indispensable garantizar que **todas las operaciones de una transacción se completen o se cancelen** en su totalidad.

## B. Comercio electrónico

- Cuando un cliente realiza una compra, se ejecutan múltiples operaciones: validación de stock, registro del pedido, actualización de inventario, y generación de comprobante.
- Si una de estas operaciones falla, la transacción debe **revertirse completamente** para evitar inconsistencias.

## C. Sistemas de salud

- Las historias clínicas, los diagnósticos y los tratamientos deben actualizarse de forma confiable.
- Varias personas pueden acceder simultáneamente a los datos del mismo paciente, por lo que se requiere un **control de concurrencia efectivo** y transacciones duraderas.

## Conclusión General

El diseño y uso correcto de bases de datos en sistemas modernos requiere mucho más que almacenar datos: implica **garantizar su validez, coherencia, disponibilidad y seguridad** ante múltiples accesos simultáneos. En este marco, MySQL (especialmente con el motor InnoDB) proporciona las herramientas necesarias para construir aplicaciones confiables en entornos transaccionales.

El concepto de **integridad** permite definir reglas que aseguran que los datos sean válidos, desde las claves primarias y foráneas hasta restricciones de tipo y reglas de negocio. La **consistencia** garantiza que cualquier cambio en la base respete esas reglas, manteniendo siempre un estado coherente. La **disponibilidad**, por su parte, se logra mediante estrategias como replicación, respaldo, monitoreo y configuración de clusters que permiten que el sistema esté siempre accesible.

Las **transacciones** son unidades lógicas de trabajo que aseguran que las operaciones se ejecuten de manera completa, cumpliendo con las propiedades **ACID**: atomicidad, consistencia, aislamiento y durabilidad. Estas propiedades son clave en aplicaciones críticas como sistemas bancarios, plataformas de comercio electrónico y registros médicos.

La **concurrencia**, inevitable en sistemas multiusuario, se gestiona eficazmente en MySQL a través de mecanismos como **MVCC, bloqueos a nivel de fila y niveles de aislamiento configurables**, que permiten equilibrar rendimiento con integridad. Además, el reconocimiento automático de **deadlocks** y la posibilidad de implementar enfoques **optimistas o pesimistas** de control, brindan flexibilidad para distintos tipos de aplicaciones.

En síntesis, al comprender y aplicar correctamente estos principios, es posible diseñar bases de datos robustas, preparadas para trabajar en contextos complejos y exigentes, minimizando riesgos y garantizando la confiabilidad de la información en todo momento.