TRANSACCIONES

- Supongamos que tenemos una tabla como esta: CUENTA (Id_Cuenta, Saldo).
- Ahora, supongamos que queremos enviar dinero de una cuenta a otra. Es decir, tenemos la cuenta (1000, 14890) de la que queremos extraer 500€ y enviarlos a la cuenta (2000, 3080). Es decir:

1000	14890
2000	3080

1000	14390
2000	3580

- ¿Qué instrucciones SQL necesitamos?

```
1→ UPDATE CUENTA SET Saldo=Saldo - 500  WHERE Id_Cuenta=1000;

2→ UPDATE CUENTA SET Saldo=Saldo + 500 WHERE Id_Cuenta=2000;
```

- Ahora vamos a suponer que ejecutamos la instrucción 1 y justo después ocurre algo como:
  - Disco, memoria, CPU,... algo peta
  - Se va la luz.
  - Perdemos la conexión.
  - Nos llaman y pasamos de ejecutar la 2 porque después de atender la llamada nos vamos a almorzar
  - …
- O al ejecutar la instrucción 2 sucede:
  - Se produce un error: la cuenta 2000 no existe (la han eliminado mientras ejecutábamos la 1)
  - No tenemos permisos para cambiar la cuenta 2000 (nos han cambiado los permisos justo después de modificar la cuenta 1000)
  - Alguien ha eliminado la tabla CUENTA
  - Alguien ha modificado el nombre del campo Saldo
  - ...

- Ante todas estas eventualidades la solución es clara → debemos devolver a la cuenta 1000 los 500€ que no hemos podido añadir en la cuenta 2000. Es decir, deshacer el cambio producido por la instrucción 1.
- Ahora bien, en el tiempo que va desde que ejecutamos la instrucción 1 hasta que nos damos cuenta de que no vamos a poder ejecutar con garantías la instrucción 2, el cliente de la cuenta 1000 tiene 500€ menos de los que debiera tener…

¿Se os ocurre alguna solución?

Transacción: Definición

- Conjunto de instrucciones DML que son tratadas como una única unidad.
- La transacción será válida y por tanto ejecutada, si todas las instrucciones que agrupa lo son. Si cualquiera de dichas instrucciones falla la transacción se cancela y se dejan los datos tal y como estaban al inicio de la misma.
- Una transacción por tanto debe cumplir el paradigma ACID (Atomicity, Consistency, Isolation, Durability):
    - **Atomicity**: todas las instrucciones que forman la transacción se ejecutan con éxito o todas son canceladas.
    - **Consistency**: la bbdd que inicia una transacción en un estado consistente debe acabar en otro estado consistente.
    - **Isolation**: una transacción no debe afectar a otra.
    - **Durability**: los cambios realizados por una transacción son persistentes. Ningún cambio se pierde.

- Usos:
  - Para agrupar un conjunto de instrucciones
  - Cuando varios clientes accederán a los mismos datos de forma concurrente.
- Consideraciones
  - Una transacción consume muchísimos más recursos que una operación normal en la base de datos. Se deben utilizar solo en casos de necesidad.
  - No todos los sistemas admiten transacciones (son transaccionales). MySql solo trabaja de forma transaccional con InnoDB. Para ver qué motores de MySql son transaccionales podemos hacer: `SHOW ENGINES`.

Instrucciones de control de una transacción

- **START TRANSACTION (o BEGIN)**: inicia explícitamente una transacción
- **SAVEPOINT**: crea un punto de referencia al que podemos volver
- **COMMIT**: compromete los cambios realizados.
- **ROLLBACK**: cancela los cambios realizados.
- **ROLLBACK TO SAVEPOINT**: cancela los cambios realizados desde el punto de referencia creado con SAVEPOINT.
- **RELEASE SAVEPOINT**: libera la referencia creada por el SAVEPOINT
- **SET AUTOCOMMIT**: habilita o deshabilita el modo autocommit. Por defecto está activado

Problemas: Transacciones_1

- Piensa una situación (como la de la transacción bancaria) en la que sea necesario utilizar alguna transacción.
- Realiza una consulta al diccionario de datos para averiguar qué motores de almacenamiento admiten transacciones.
- ¿Qué dice el manual sobre las transacciones? Échale una ojeada.
- Comprueba que la tabla City admite transacciones.
- Empieza una transacción.
- Elimina una fila de la tabla City.
- Deshaz el cambio. ¿Qué sucede?
- Inicia una nueva transacción y vuelve a eliminar una fila de City
- Compromete los cambios y comprueba que efectivamente ha desaparecido la fila, ¿puedes deshacer los cambios?

Modo AUTOCOMMIT

- Determina cuándo y cómo se realiza la transacción.
- Trata todas las instrucciones que lo requieran como una transacción. Es decir, obliga a toda instrucción DML a ser tratada como una transacción. Si la instrucción se ejecuta con éxito, compromete el resultado, si no, cancela todos los cambios producidos por la instrucción.
- Para deshabilitar este modo hay varias opciones:
  - set autocommit=0;
  - set session autocommit=0;
  - set @@autocommit :=0;
- Si el modo autocommit está deshabilitado:
  - Cada transacción abarca varias instrucciones.
  - Podemos finalizar una transacción utilizando COMMIT o ROLLBACK.
- Para visualizar el modo actual de la variable autocommit:

```
select @@autocommit;
```

COMMIT implícito

- Algunas instrucciones, llamadas generalmente "instrucciones no transaccionales", provocan la ejecución de un COMMIT implícito. Esto es, el sistema ejecuta un COMMIT antes de ejecutar la transacción. La razón está en que estas instrucciones no son reversibles, es decir, no admiten ROLLBACK.
- Ejemplo de instrucciones que provocan un COMMIT implícito y, por tanto, que la transacción concluya:
  - START TRANSACTION
  - SET AUTOCOMMIT
  - DDL: ALTER, CREATE, DROP…
  - DCL: GRANT, REVOKE…
  - Otras: LOCK TABLES, LOAD DATA INFILE, TRUNCATE TABLE....

Problemas con el grado de aislamiento de las transacciones

- Dirty read: cuando una transacción lee cambios cometidos por una transacción que no ha sido comprometida. Ejemplo: transacción T1 realiza una serie de cambios. T2 lee dichos cambios. T1 finalmente decide no comprometer dichos cambios y ejecuta un ROLLBACK. T2, por tanto, trabajará con datos erróneos.
- Non-repeatable read: cuando una lectura de datos no puede ser repetida dentro de una misma transacción porque otra transacción los ha eliminado.
- Phantom read: cuando aparece una nueva fila entre dos lecturas de la misma tabla dentro de la misma transacción. Ejemplo: T1 y T2 son dos transacciones concurrentes y T1 lee un conjunto de filas. Si T2 añade una nueva fila y, a continuación, T1 vuelve a leer obtendrá una fila nueva que antes no disponía: la fila fantasma.

Grados de aislamiento en MySql

- READ UNCOMMITTED: permite que la transacción vea los cambios no comprometidos realizados por otras transacciones. Es decir, permite los tres problemas comentados anteriormente.
- READ COMMITTED: solo permite que la transacción vea cambios comprometidos. Es decir, no permite las lecturas sucias ('dirty reads').
- REPEATABLE READ: Asegura que la transacción siempre leerá los mismos datos independientemente de si hay cambios o no durante el transcurso de la transacción. Es decir, evita tanto las lecturas sucias como el que no se puedan repetir las lecturas ('non-repeatable read'). Este es el modo por defecto de InnoDB.
- SERIALIZABLE: asegura que la transacción está completamente aislada de cualquier efecto producido por otras transacciones. Por tanto, cualquier dato consultado por una transacción serializable permanecerá invariable durante toda la transacción.

SET TRANSACTION ISOLATION LEVEL

Para establecer el grado de aislamiento de cada transacción utilizaremos SET TRANSACTION ISOLATION LEVEL. Podemos hacerlo a nivel de sesión o a nivel global. Ejemplos:

- SET GLOBAL TRANSACTION ISOLATION LEVEL READ COMMITTED; establecemos que para todas la conexiones futuras (no afecta a las actuales) el grado de aislamiento será READ COMMITTED.
- SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED; establecemos el nivel de aislamiento pero en el ámbito de la sesión actual.
- SELECT @@global.tx_isolation, @@session.tx_isolation; para consultar los valores actuales de TRANSACTION ISOLATION LEVEL.

Problemas: Transacciones_2

- Inicia una sesión y cámbiale el PROMPT a T1.
- ¿Cuál es el grado de aislamiento de las transacciones en esta sesión?
- Inicia transacción en T1.
- Consulta las filas de City con ID>4070.
- En otra ventana, inicia sesión y cámbiale el PROMPT a T2.
- Inicia transacción en T2. Consulta en T2 las filas de City con ID>4070.
- Inserta en T1 una nueva fila en City (Nueva_ciudad, ATA).
- Repite la consulta anterior en T2,¿qué sucede?
- Compromete el cambio en T1.
- Repite la consulta anterior en T2,¿qué sucede?
- Vuelve a T1 y deshaz el cambio en una nueva transacción.
- En T2 cambia el grado de aislamiento a READ UNCOMMITTED.
- Repite los pasos anteriores y observa qué sucede.