CONCURRENCIA

Aprendamos a manejar sistemas multi-usuario de forma correcta!

CONCURRENCIA

¿Cuál o cuales de las propiedades **ACID** estan relacionadas con la concurrencia?

I: Isolation. Aislamiento de las transacciones

Por defecto, las BDs relacionales estan configuradas relajando el aislamiento completo, para aumentar la performance y escalar mejor.

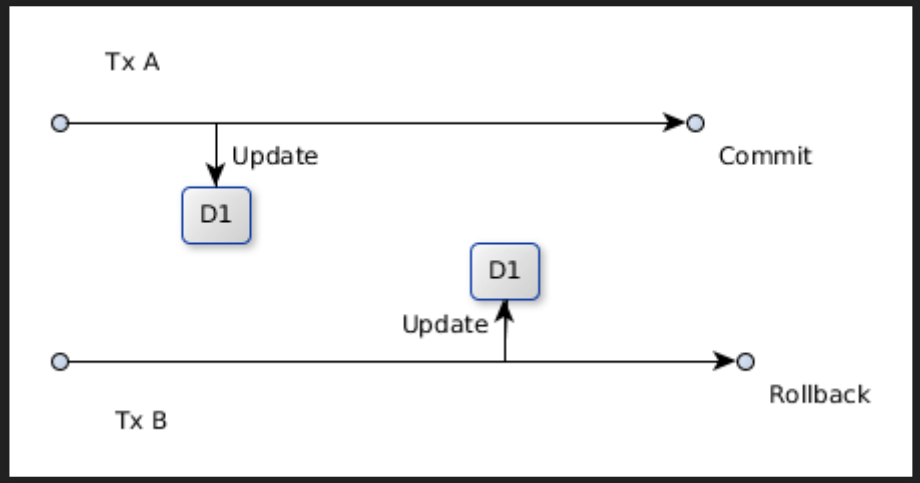
El aislamiento completo no permite ejecutar transacciones en forma concurrente.

CONCURRENCIA

¿Cuáles son los problemas que podemos encontrarnos al no configurar la BD con aislamiento completo?

CONCURRENCIA

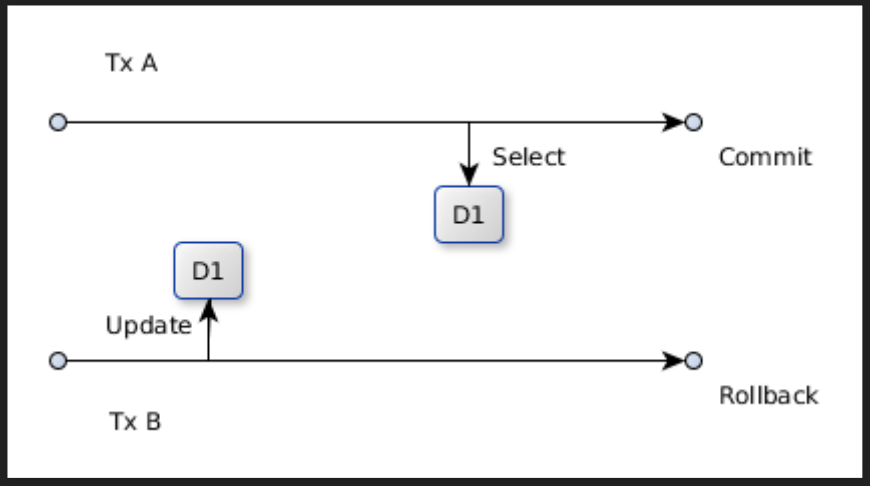
Lost Update: Sucede si dos transacciones modifican el mismo dato y la segunda transacción aborta.



TxA cambia D0 por D1, y TxB cambia D1 por D2, pero "rollbackea", volviendo a D0.

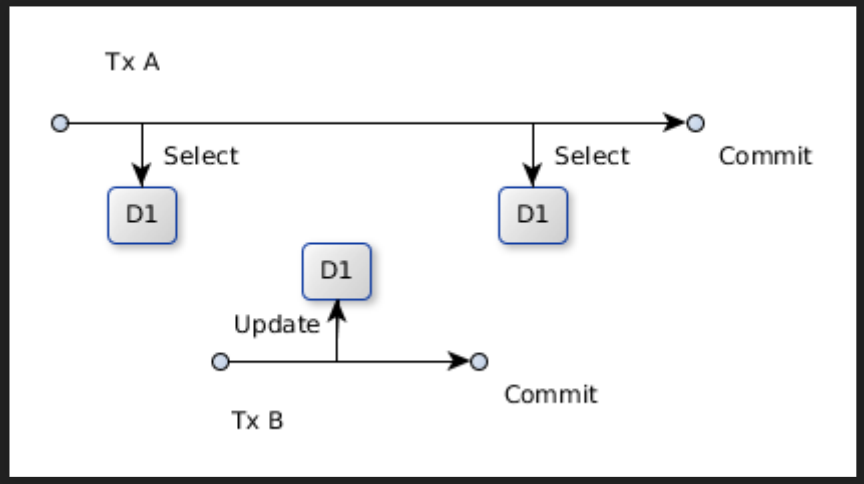
CONCURRENCIA

Dirty Read: Sucede si una transacción lee datos modificados por otra transacción que aún no comiteo. Esto es peligroso porque podría darse que la transacción termine abortado, con lo cual se leyeron datos espurios/sucios.



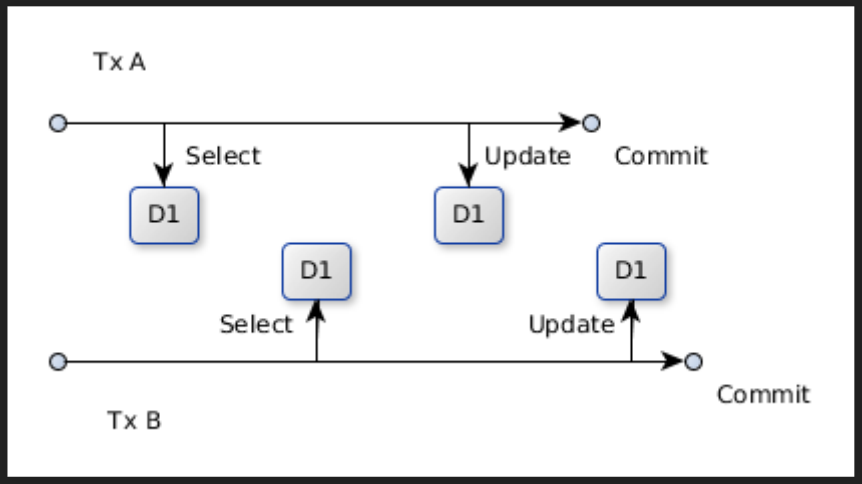
CONCURRENCIA

Unrepeatable Read: Sucede si una transacción lee un dato dos veces y la segunda vez es diferente a la primera.



CONCURRENCIA

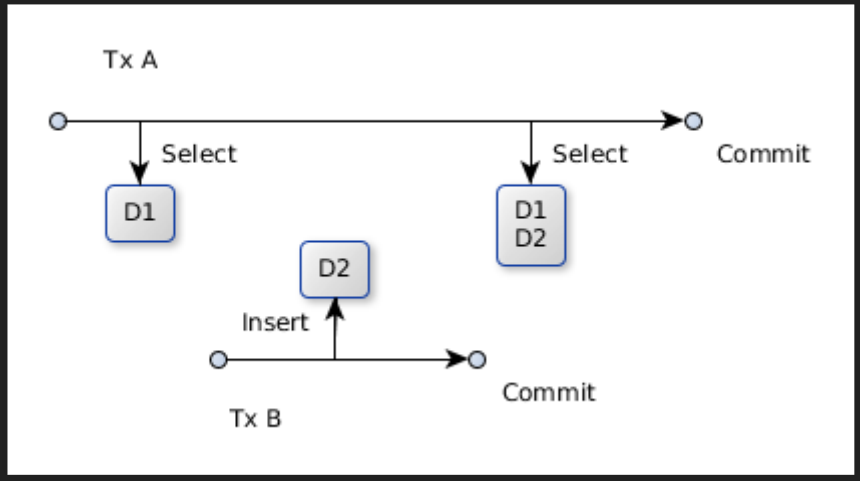
Last Commit Wins: Dos transacciones leen el mismo dato, la primera lo modifica y comitea y la segunda hace lo mismo después.



Esto es muy frustrante para los usuarios, ya que el primero pierde los cambios que hizo.

CONCURRENCIA

Phanton Read: Ocurre cuando una transacción ejecuta una query dos veces, y la segunda vez obtiene mas datos (o menos si se borraron) que la primera.

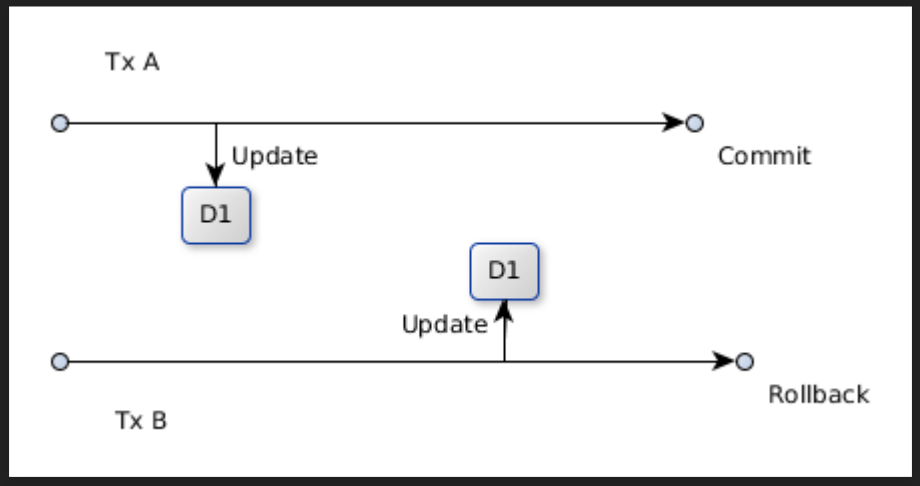


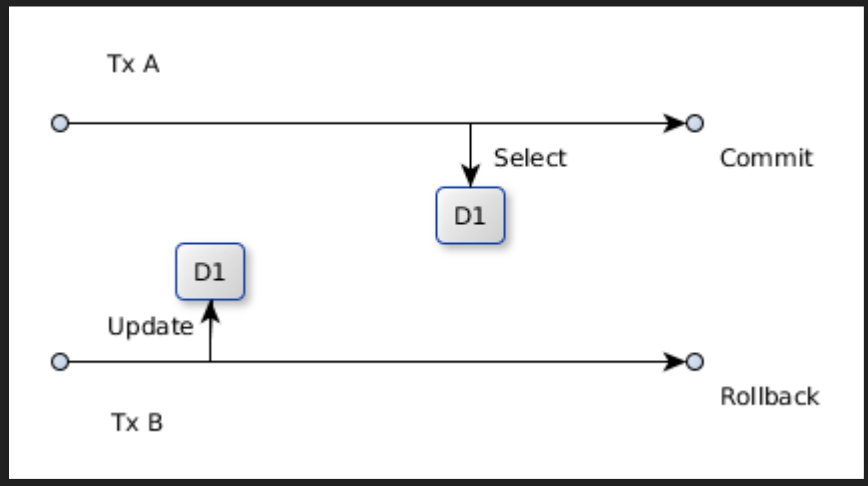
ANSI SQL ISOLATION LEVELS

¿Niveles de Aislamiento? ¿Qué son?

ANSI SQL ISOLATION LEVELS

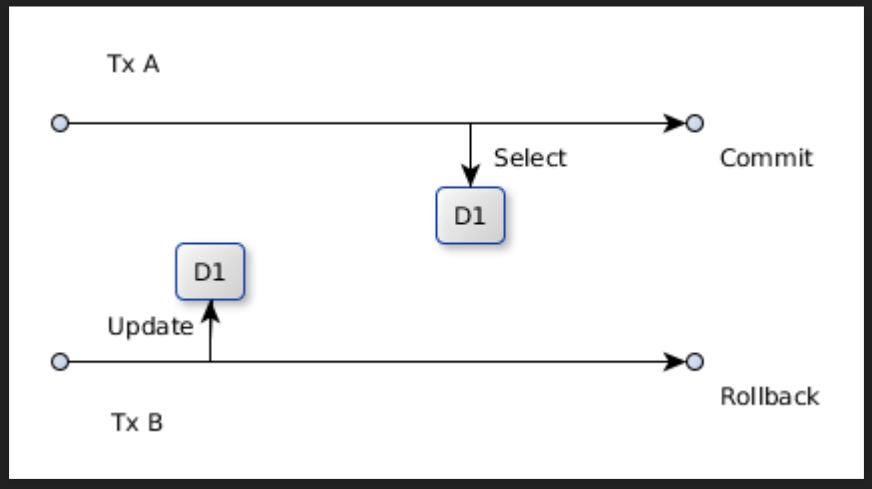
1. Read Uncommited: Permite dirty-reads pero no permite lost-updates. O sea, una transacción no tiene permitido escribir un dato que fue modificado por otra transacción que aún esta ejecutándose. Pero si puede leer cualquier dato.

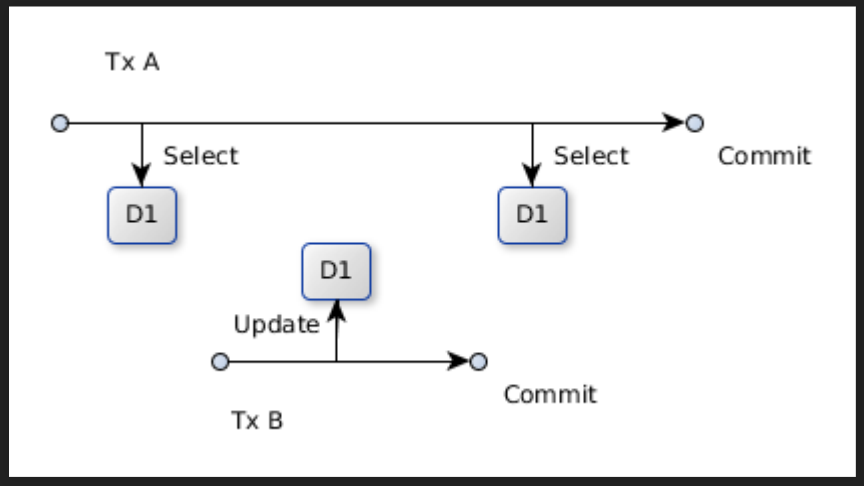




ANSI SQL ISOLATION LEVELS

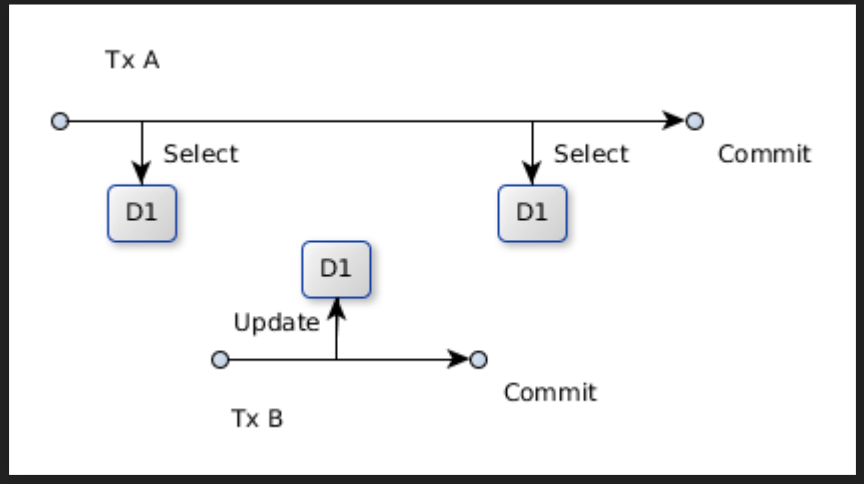
2. Read Commited: Permite unrepeatable reads pero no permite dirty reads. Las lecturas dentro de la transacción no bloquean, pero las escrituras (el where del update) bloquean cualquier tipo de lectura/escritura de otras transacciones concurrentes.

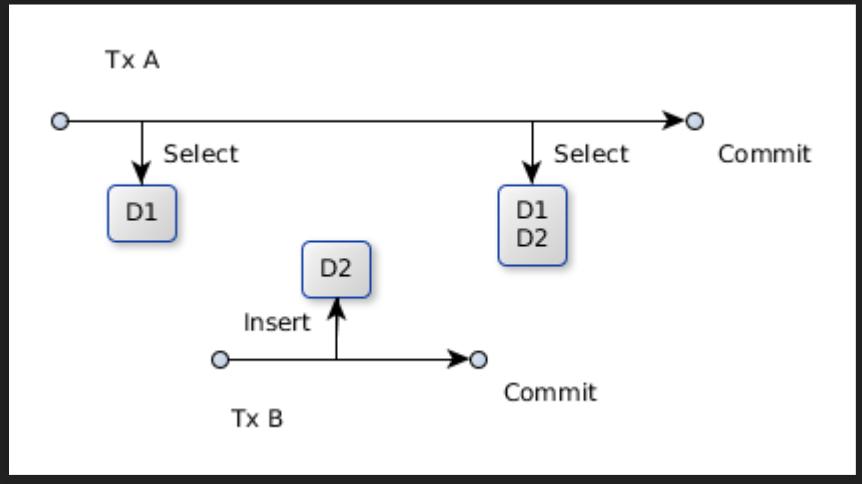




ANSI SQL ISOLATION LEVELS

3. Repeatable Read: No permite unrepeatable read, si permite phanton reads. Las lecturas dentro de una transacción bloquean las escrituras (el where de un update por ejemplo) de otras transacciones concurrentes.





ANSI SQL ISOLATION LEVELS

4. Serializable: No permite phanton reads. Una transacción se ejecuta despues de otra (en serie). No hay concurrencia alguna.

CONCURRENCIA

ISOLATION LEVELS (REPASO)

1. Read Uncommited
2. Read Commited
3. Repeatable Read
4. Serializable

¿Que Base de Datos estan usando? ¿Saben en que nivel de aislamiento esta ejecutándose?

PostgreSQL, Oracle y MS SQL Server por defecto usan Read Committed. MySQL y MariaDB en Repeatable Read.

CONCURRENCIA

ALGUNAS REFLEXIONES

Read Uncommitted es muy peligroso, por eso por defecto debe descartarse.

Serializable es muy costoso en términos de escalabilidad del sistema y las lecturas fantasmas no suelen ser un problema en la mayoría de los casos. Sí es probable que necesitemos serializar ciertas acciones (casos de uso) de una aplicación.

CONCURRENCIA

ALGUNAS REFLEXIONES

No hay una única forma de prevenir todos los casos particulares que pueden darse en una aplicación. Debemos manejarlos desde la aplicación, segun casos de uso.

CONCURRENCIA

LAST UPDATE WINS

¿Cuál de los niveles de ailamiento vistos lo previene?

Ninguno

Porque la lectura y modificación son en transacciones separadas!

Ej: Un usuario lee un producto con intenciones de modificarlo, otro hace lo mismo y ambos lo modifican...

Hay que resolverlo a nivel aplicación

CONCURRENCIA

LAST UPDATE WINS

Dos estratégias para resolverlo a nivel aplicación:

- Optimista
- Pesimista

Elegir una u otra depende de cuan frecuente podría darse que los usuarios se pisen entre ellos.

CONCURRENCIA

ESTRATÉGIA OPTIMISTA

Los objetos tienen un número de versión y al leer el objeto la obtienen. Al persistir una modificación primero revisan si la versión es la misma, si lo es, persisten el cambio y actualizan la versión sino lanzan una excepción.

CONCURRENCIA

ESTRATÉGIA PESIMISTA

Al leer un objeto, éste se marca en BD como "bloqueado" y queda bloqueado para el resto de los usuarios.

Se libera con la acción de persistir el cambio, o con una acción específica de "desbloquear".

Ojo!: Si un usuario lo bloquea y se va a su casa... nadie podra accederlo. Plan B!

CONCURRENCIA

PROBLEMAS QUE RESUELVO CON LA BASE DE DATOS

Ejemplo: Crear un expediente que debe generarse con un número único que se resetea cada nuevo año.

```
Tx begin
 n = leer numero actual para año en curso
 n = n + 1;
 update n;
Tx commit
```

¿Qué nivel de aislamiento lo previene?

Leamos utilizando "select for update"

JAVA PERSISTENCE API (JPA)

ESTRATEGIA OPTIMISTA EN JPA

```
@Entity
public class Persona {
 ...
 @Version
 private Long version;
 ...
}
```

Definir la versión y luego manejar la excepción *javax.persistence.OptimisticLockException*