CONCURRENCIA

Aprendamos a manejar sistemas multi-usuario de forma correcta!

CONCURRENCIA

Problema 1: Compra de asientos de colectivo para el mismo viaje desde dos sucursales distintas.

El empleado le muestra la pantalla al comprador con los asientos disponibles. Supongamos se elije el asiento 10 y confirma:

```
Tx begin
  ocupado = select 1 from asientos
                        where id_asiendo = 10 and ocupado = 1 and viaje = 2
  if (ocupado)
    throw new Exception("Lo siento, el asiento fue vendido");
  update asientos set ocupado = 1
                  where id_asiendo = 10 and ocupado = 0 and viaje = 2
  insert into asiento_vendido (10 (asiento), 2 (viaje),  comprador_x)
Tx commit
```

CONCURRENCIA

¿Le voy a vender el mismo asiento a dos personas distintas o no?

CONCURRENCIA

Lo resolvemos utilizando "select for update"

```
Tx begin
  ocupado = select 1 from asientos
                where id_asiendo = 10 and ocupado = 1 and viaje = 2
                for update
  if (ocupado)
    throw new Exception("Lo siento, el asiento fue vendido");
  update asientos set ocupado = 1
                  where id_asiendo = 10 and ocupado = 0 and viaje = 2
  insert into asiento_vendido (10 (asiento), 2 (viaje),  comprador_x)
Tx commit
```

CONCURRENCIA

PROBLEMA 2

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

```
Tx begin
  n = select numero from numero_exp where año = AAAA;
  n = n + 1;
  -- n para el nuevo expediente
  update numero_exp set numero = n where año = AAAA;
Tx commit
```

CONCURRENCIA

PROBLEMA 2

Usemos "select for update" nuevamente

```
Tx begin
  n = select numero from numero_exp where año = AAAA for update
  n = n + 1;
  -- n para el nuevo expediente
  update numero_exp set numero = n where año = AAAA;
Tx commit
```

CONCURRENCIA

PROBLEMA 3 - LAST UPDATE WINS

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

El usuario se paso 10 minutos modificando el producto y luego pierde sus cambios...

¿Cómo lo resuelvo de forma elegante?

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!

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*