01-2025

code URJC

Desarrollo Web

Tecnologías de servidor web

Tema 2.7: Concurrencia

Universidad Rey Juan Carlos

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto e Iván Chicano

code
URJC

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto, Iván Chicano

Accesos concurrentes

code URJC

- Es importante gestionar de forma adecuada qué ocurre cuando varios usuarios interactúan con una aplicación con base de datos **concurrentemente**
- Cuando los usuarios únicamente consultan información **no hay interferencias entre ellos**
- Cuando un usuario (o los dos) escriben, se pueden producir **interferencias no deseadas**

Accesos concurrentes

- **Ejemplos de situaciones problemáticas:**
  - Dos usuarios consultan un recurso desde la web a la misma vez.
    - Uno de ellos modifica una propiedad y reemplaza el recurso.
    - El otro modifica otra propiedad y también reemplaza el recurso borrando los cambios de la modificación previa (*lost updates*)
  - Ejemplo: Dos usuarios deciden comprar la última entrada de un concierto a la misma vez

Accesos concurrentes

code URJC

- **Existen varias técnicas que nos ayudan a gestionar estas situaciones**
  - Transacciones de la base de datos
  - Cabeceras ETag e If-Match en la API REST
  - Sentencias SQL que verifican precondiciones

Accesos concurrentes

code URJC

- **Transacciones de la base de datos**
  - Varias sentencias se ejecutan dentro de una unidad lógica
  - Los cambios de todas ellas se aplican o se descartan. No es posible aplicar cambios parciales

https://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html

https://docs.spring.io/spring-framework/docs/current/reference/html/data-access.html#transaction

Accesos concurrentes

- **Transacciones de la base de datos**
  - El grado de aislamiendo de las transacciones es configurable (*Isolation level*)
  - Afecta a los datos que se pueden leer desde una transacción de otras transacciones que se ejecutan de forma concurrente.

Transacciones de las base de datos

code URJC

- **Isolation level**
  - **READ_UNCOMMITED:** Lectura de valores escritos por otras transacciones que todavía no han finalizado (y podrían abortarse finalmente) (*Dirty reads*)
  - **READ_COMMITED:** Si durante la transacción se lee la misma fila dos veces y esa fila es modificada por otra transacción entre medias se leen valores diferentes (*Non-repeatable reads*)
  - **REPEATABLE_READ:** Si se hace una consulta y se obtienen unas filas y otra transacción añade nuevas filas y se vuelve a hacer la misma consulta, aparecen filas nuevas (*Phantom reads)*
  - ***SERIALIZABLE:*** Las transacciones están aisladas entre sí. Su comportamiento es como si se ejecutaran secuencialmente.

Transacciones de las base de datos

code URJC

- Isolation level

Isolation level	Lost updates	Dirty reads	Non-repeatable reads	Phantoms
Read Uncommitted	don't occur	may occur	may occur	may occur
Read Committed	don't occur	don't occur	may occur	may occur
Repeatable Read	don't occur	don't occur	don't occur	may occur
Serializable	don't occur	don't occur	don't occur	don't occur

Fuente: https://alxibra.medium.com/isolation-level-in-rails-847edc9e347d

Accesos concurrentes

- **Cabeceras ETag e If-Match en la API REST**
  - **Problemática (*lost updates*):**
    - Dos usuarios consultan un recurso desde la web a la misma vez.
    - Uno de ellos modifica una propiedad y reemplaza el recurso.
    - El otro modifica otra propiedad y también reemplaza el recurso (borrando los cambios de la modificacion previa)

https://www.baeldung.com/etags-for-rest-with-spring
https://dzone.com/articles/concurrency-control-in-rest-api-with-spring-framew

Accesos concurrentes

- **Cabeceras ETag e If-Match en la API REST**
  - **Solución: *Optimistic locking***
    - Cada recurso tiene asociada una versión
    - Cuando se lee un recurso se recibe la versión (**cabecera ETag**)
    - Cuando se quiere modificar el recurso (PUT o PATH) se tiene que enviar la versión recibida previamente (**cabecera If-Match**)
    - Si el recurso no ha sido modificado y sigue en esa versión, se acepta el cambio. Si no, error.

Accesos concurrentes

- **Sentencias SQL que verifican precondiciones**
  - **Problemática:**
    - Dos usuarios deciden comprar la última entrada de un concierto a la misma vez
    - Sólo uno de ellos debería poder comprar la entrada

Accesos concurrentes

- **Sentencias SQL que verifican precondiciones**
  - **Solución:**
    - Se podría usar un nivel de aislamiento de las transacciones SERIALIZABLE. Eso implica que la primera transacción consigue la entrada y la segunda detecta que ya no hay disponibles.
    - También se pueden ejecutar sentencias SQL atómicas que tengan la precondición de que queden plazas libres (más eficiente)

Accesos concurrentes

code URJC

- Sentencias SQL que verifican precondiciones

```
@Transactional
@Modifying
@Query("UPDATE Event e SET e.tickets=e.tickets+1 " +
       "WHERE e.id = :id AND e.tickets+1 <= e.max_capacity")
public int reserveTicket(@Param("id") long id);
```

```
int rowsUpdated = eventRepository.reserveTicket(eventId);
if (rowsUpdated == 1) {
    //Ticket buy
} else {
    //Event full. No ticket
}
```