# Übung: Locking in JPA

## Pessimistisches Locking

Pessimistisches Locking geht von der Annahme aus, dass es höchstwahrscheinlich zu Konflikten im Datenzugriff kommen wird. Um diese zu vermeiden, können Daten für weitere Zugriffe gesperrt werden (Lock).

Grundsätzlich stehen 2 Arten von Locks zur Verfügung:

**SHARED LOCK - `PESSIMISTIC_READ`**

Daten können von anderen gelesen werden, aber NICHT verändert

**EXCLUSIVE LOCK - `PESSIMISTIC_WRITE`**

Daten können nur von der Transaktion geändert werden, die das Exclusive-Lock besitzt

## Locks setzen

In der Klasse `boundary.PessimisticResource` finden Sie 2 Methoden `read()` and `write`.

Diese führen zu Demonstrationszwecken jeweils eine Datenbankabfrage aus und wartet dann 30 Sekunden, um eine längere Bearbeitungszeit zu simulieren.

Auf der Konsole wird jeweils eine Meldung ausgegeben, sobald die Query ausgeführt wurde.

### Aufruf ohne Locks

Starten Sie einen Browser und öffnen Sie in unterschiedlichen Tabs folgende URLs:

- http://localhost:8080/pessimistic/write/Sonne
- http://localhost:8080/pessimistic/write/Huber

Achten Sie auf die Konsolenausgabe, es müsste jeweils zeitnah die Ausgabe der Person erfolgen. Im Browser werden die Daten 30 Sekunden verzögert geliefert.

### Lock setzen auf die Query

Ergänzen Sie in der Methode `write()` folgende Zeile:

```
query.setLockMode(LockModeType.PESSIMISTIC_WRITE);
```

Wiederholen Sie die Aufrufe und achten Sie wieder auf die Konsolenausgabe.

Die erste Ausgabe erfolgt sofort, die zweite Ausgabe erfolgt erst, nachdem die erste Transaktion abgeschlossen wurde und somit das Lock wieder freigegeben wurde. Erst dann kann der zweite Aufruf das Lock erhalten.

Versuchen Sie auch, die Methode `read()` aufzurufen. Diese wird sofort ein Ergebnis liefern, weil darin nicht explizit ein Lock gesetzt wird.

## Lock setzen auf Datensatz

Im vorherigen Ansatz war zu sehen, dass nicht nur der betroffene Datensatz gesperrt wurde.

Wir können das Lock auch auf einzelne Datensätze anwenden. Passen Sie dazu den Sourcecode wie folgt an:

```
Person p = query.getSingleResult();

// or lock only the choosen Person
entityManager.lock(p, LockModeType.PESSIMISTIC_WRITE);
```

Versuchen Sie dann nochmals die `write()`-Aufrufe. Beide Aufrufe liefern sofort ein Ergebnis auf der Konsole, sofern sie nicht den selben Datensatz betreffen.

Ergänzen Sie nun das Lock auf den jeweiligen Ergebnis-Datensatzes der `read()`-Methode:

```
Person p = query.getSingleResult();

entityManager.lock(p, LockModeType.PESSIMISTIC_READ);
logger.info("PESSIMISTIC_READ: " + p);
```

Starten Sie dann zuerst den `write`-Aufruf, und dann unmittelbar im Anschluss den `read`-Aufruf.+ Der Lesevorgang müsste nun blockiert werden, bis das exklusive Lock freigegeben wurde.

Derby-DB weicht hier bei den Locks etwas ab, indem neben dem exklusivem Lock noch ein UPDATE-Lock Verwendung findet. Dieses erlaubt die Vergabe eines Shared-Locks…

Weitere Infos zum pessimistischen Locking in JPA finden Sie unter anderem auf https://www.baeldung.com/jpa-pessimistic-locking.

# Optimistic Locking

Beim optimistischen Locking geht man grundsätzlich davon aus, dass keine/wenige Konflikte auftreten werden. Die Datensätze werden nicht vorab gesperrt, was die Performance erhöhen kann. Im Gegensatz dazu wird vor dem Festschreiben der Daten geprüft, ob diese zwischenzeitlich verändert wurden. Dies kann über Zeitstempel der letzten Änderung oder einfach über ein VersionsNr-Attribut (JPA) umgesetzt werden.

Optimistisches Locking ist demnach von Vorteil, wenn mit vielen Leseoperationen, aber wenig Schreiboperationen zu rechnen ist.

Folgende Locks stellt JPA dafür bereit:

`OPTIMISTIC` **(früher** `READ`**)**

Schützt vor Dirty-Reads und Non-Repeatable-Reads

`OPTIMISTIC_FORCE_INCREMENT` **(früher** `WRITE`**)**

Erhöht jeweils den Wert des Version-Attributs.

Für das Version-Attribut sind folgende Datentypen zulässig:

- int, Integer, long, Long, short, Short
- java.sql.Timestamp

## Entity um `@Version`-Attribut erweitert

Ergänzen Sie die Klasse `Person` um ein @Version-Attribut:

```
@Version ①
private Long version;

// Getter and Setter
```

① Attribut ist bereits vorhanden, wird aber erst durch die @Version-Annotation entsprechend benutzt!

## `OPTIMISTIC`-Lock setzen

Ergänzen Sie in der `OptimisticResource` in der Methode `read()` das entsprechende Lock:

```
query.setLockMode(LockModeType.OPTIMISTIC);
```

Versuchen Sie mehrere READ-Zugriffe hintereinander und achten Sie wieder auf die Log-Ausgabe. Es sollste zu keinen Verzögerungen kommen.

## `OPTIMISTIC_FORCE_INCREMENT`-Lock setzen

Ergänzen Sie die `write()`-Methode in der Klasse `OptimisticResource` wie folgt:

```
query.setLockMode(LockModeType.OPTIMISTIC_FORCE_INCREMENT);
```

Rufen Sie nun die `write()`-Methode auf und unmittelbar danach die `read()`-Methode.

Der Aufruf der `write()`-Methode erhöht den Versionszähler der betroffenen Person.
Beim Transaktionsende der `read()`-Methode wird festgestellt, dass sich die Version im Laufe der Transaktion verändert hat, und die Daten somit nicht mehr aktuell wären. Aus diesem Grund wird eine entsprechende Exception geworfen:

```
Caused by: org.hibernate.dialect.lock.OptimisticEntityLockException: Newer version [3]
of entity [[at.htl.model.Person#3]] found in database
```