Applications orientées données (NSY135)

14 – Applications concurrentes

Auteurs: Raphaël Fournier-S'niehotta et Philippe Rigaux
(philippe.rigaux@cnam.fr,fournier@cnam.fr)

Département d'informatique
Conservatoire National des Arts & Métiers, Paris, France

Plan du cours

Gestion des niveaux d'isolation

- Hibernate s'appuie sur le niveau d'isolation fourni par le SGBD, et ne tente pas de ré-implanter des protocoles de concurrence.
- Le mode par défaut dans la plupart des systèmes est **read committed** ou **repeatable read**.
- Ce niveau d'isolation (par défaut) n'offre pas toutes les garanties
- il est indipensable de se poser sérieusement la question des risques liés à la concurrence d'accès.
- Dans une application transactionnelle, le mode **read uncommitted** ne devrait même pas être envisagé.
- Il reste donc comme possibilités :
  - d'accepter le mode par défaut, après évaluation des risques;
  - ou de passer en mode **serializable**, en acceptant les conséquences (performances moindres, risques de **deadlock**);
  - ou, enfin, d'introduire une dose de gestion "manuelle" de la concurrence en effectuant des verrouillages préventifs (dits, parfois, "verrouillage pessimiste").

Gestion des niveaux d'isolation

- La configuration Hibernate permet de spécifier le mode d'isolation choisi pour une application :

```
1        hibernate.connection.isolation = <val>
```

- **val** est un des codes suivants :
  - 1 pour **read uncommitted**
  - 2 pour **read committed**
  - 3 pour **repeatable read**
  - 4 pour **serializable**
- On peut donc spécifier un niveau d'isolation, qui s'applique à **toutes** les sessions, même celles qui ne présentent pas de risque transactionnel.
- Dans ces conditions, choisir le niveau maximal (**serializable**) systématiquement représente une pénalité certaine.
- Il semble préférable d'utiliser le niveau d'isolation par défaut du SGBD, et de changer le mode d'isolation à **serializable** ponctuellement pour les transactions sensibles.

Niveaux d'isolation

- Il ne semble malheureusement pas possible, avec Hibernate, d'affecter simplement le niveau d'isolation pour une session.
- On en est donc réduit à passer par l'objet **Connexion** de JDBC.

```
1       session.doWork(
2           new Work() {
3                   @Override
4                   public void execute(Connection connection) throws SQLException {
5                       connection.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
6                   }
7               });
```

- **Le mode serializable, malgré ses inconvénients (apparition d'interblocages) est le plus sûr pour garantir l'apparition d'incohérences dans la base**, dont la cause est très difficile à identifier.
- Une autre solution consiste à verrouiller explicitement, au moment de leur mise en cache, les objets que l'on va modifier.

Verrouillage avec Hibernate

- Hibernate se contente de transmettre les requêtes de verrouillage au SGBD
- aucun verrou en mémoire n'est posé
- (cela n'a pas de sens car chaque application ayant son propre cache, un verrou n'aurait aucun effet concurrentiel).
- La principale utilité des verrous est de gérer le cas de la **lecture** d'une ligne/objet, suivie de **l'écriture** de cette ligne.
- Par défaut, la lecture pose un verrou **partagé** qui n'empêche pas d'autres transactions de lire à leur tour.

Verrouillage avec Hibernate

- Le principe du verrouillage explicite est donc d'effectuer une lecture qui **anticipe** l'écriture qui va suivre.
- C'est l'effet de la clause **for update**.

```
select * from xxx where .... for update
```

- Le **for update** déclare que les lignes sélectionnées vont être modifiées ensuite.
- Pour éviter qu'une autre transaction n'accède à la ligne entre la lecture et l'écriture, le système va alors poser un **verrou exclusif**.
- Le risque de mise à jour perdue disparaît.

Verrouillage avec Hibernate

- Avec Hibernate, l'équivalent du **for update** est la pose d'un verrou au moment de la lecture.

```
Transaction tx = session.beginTransaction();
Film film = session.get (Film.class, filmId, LockMode.UPGRADE);

film.titre = ... ; // Mises à jour

tx.commit();
```

Verrouillage avec Hibernate

- L'énumération **LockMode** implique l'envoi d'une requête au SGBD avec une demande de verrouillage. Les valeurs possibles sont :
  - **LockMode.NONE**. Pas d'accès à la base pour verrouiller (mode par défaut).
  - **LockMode.READ**. Lecture pour vérifier que l'objet en cache est synchronisé avec la base.
  - **LockMode.UPGRADE**. Pose d'un verrou exclusif sur la ligne.
  - **LockMode.WRITE**. Utilisé en interne.
- Gérer la concurrence dans le code d'une application est une opération lourde et peu fiable.
- Se limiter au principe simple suivant : quand on lit une ligne que l'on va modifier ensuite, on place un verrou avec **LockMode.UPGRADE**.
- Pour tous les autres cas, n'utilisez pas les autres modes et laissez le SGBD appliquer son protocole de concurrence.