Persistance objet

JDBC & Transactions

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 10 mars 2016

LAMSADE
UMR CNRS 7243
laboratoire d'analyse et modélisation de systèmes pour l'aide à la décision

DAUPHINE
UNIVERSITÉ PARIS

PSL
RESEARCH
UNIVERSITY

Introduction

- BD : modèle relationnel typiquement
- JDBC : accès via Java, modèle relationnel
- JPA : accès via Java, modèle objet
- JPA implémente un *ORM* : Object-Relational Mapping

JPA

- Avec Java EE, JPA généralement utilisé
- JPA s'appuie sur JDBC

Dans ce cours : JDBC puis JPA

Modèle relationnel

- JDBC permet d'accéder aux BD
- Suit le modèle relationnel
- Élément central du modèle : la relation (une table, par exemple)
- Relations produites par des JOIN, SELECT, etc.

Forces du modèle relationnel

Modèle relationnel

- JDBC permet d'accéder aux BD
- Suit le modèle relationnel
- Élément central du modèle : la relation (une table, par exemple)
- Relations produites par des JOIN, SELECT, etc.

Forces du modèle relationnel

- Garanties théoriques (algèbre relationnelle)
- Efficace
- Standard de fait depuis ~1990
- Robuste : Codd 1970, ANSI (puis ISO) SQL 1987 ; … ; 2011

(Bémol : nombreuses variations propriétaires)

Vue d'ensemble de l'API JDBC

- Fournisseur de SGBD implémente un pilote JDBC
- Obtenir une `Connection` (communique avec le pilote)
- `Connection` permet les *transactions*
- Transaction : ensemble atomique de « statements » SQL
- Gérer début et fin de transaction via la connexion
- Exécuter des statements SQL via cette connexion
- Par défaut, mode auto-commit : une transaction par stmt
- Via `Connection` : exécution requêtes, navigation de `ResultSet`s, ...
- Puis *fermer* la connexion

Cf. tutoriel

Instanciation

Approche 1 (Java SE, typiquement)

- Une classe fournisseur implémente `Driver`
- Développeur appelle `DriverManager`
- `DriverManager` trouve le pilote et l'instancie
- Exemple : `DriverManager.getConnection(url)`

Approche 2 (Java EE, typiquement)

- Une classe fournisseur implémente `DataSource`
- Source accessible via JNDI à un endroit convenu
- Développeur instancie DataSource par `lookup` JNDI puis appelle `source.getConnection(…)`

Injection de la `DataSource` (Java EE)

- Injection de ressources via `@Resource`
- Le conteneur va chercher la ressource via JNDI
- Nom JNDI par défaut selon type de la ressource
- Pour nous : `@Resource DataSource myDataSource;`

Statement et ResultSet

- Création d'un `Statement` (via `Connection`)
- Via `Statement` : exécution d'une commande SQL (SELECT, UPDATE…)
- Via `Statement` : paramétrisation possible (nb résultats max…)
- Obtention (si SELECT) d'un `ResultSet`
- `ResultSet` associé à une ligne courante ; initialement : avant la première
- Naviguer via `next()` aux lignes suivantes
- Invoquer `getInt(columnLabel)`, `getString(columnLabel)`…

PreparedStatement

- `PreparedStatement` : précompilé + paramétrisation facile
- La commande SQL contient des ?
- Invoquer `setInt`, `setString`... pour les paramètres

Exemple PreparedStatement

```
String s = "update USER set NAME = ? where ID = ?";
PreparedStatement stmt = con.prepareStatement(s);
stmt.setString(1, "NewName");
stmt.setInt(2, 1234);
boolean isResultSet = stmt.execute();
assert(!isResultSet);
assert(stmt.getUpdateCount() == 1);
```

Utiliser `PreparedStatement` pour éviter les attaques de type injection SQL !

Transactions

Par défaut, mode *auto commit* : une transaction par commande

Gestion de transactions explicite

- Invoquer `setAutoCommit` sur `Connection`
- Exécuter les commandes normalement
- Puis invoquer `commit` sur `Connection`
- Ou : `rollback`
- Voir aussi : `getTransactionIsolation`, `setTransactionIsolation`

PostgreSQL

- Installer PostgreSQL (site ou `sudo apt-get install postgresql`)
- Possible d'utiliser l'interface graphique d'administration pgAdmin voir logs si nécessaire
- Instructions ci-dessous pour ligne de commande linux, adapter pour autres OS
- En mode privilégié : `sudo -u postgres bash`
  - Se créer un utilisateur avec mot de passe : `createuser -P user`
  - Créer une base de données à laquelle cet utilisateur a accès : `createdb -O user db`
- Test connexion : psql db (ok sans mot de passe)
- Test connexion réseau : psql -h localhost db (exige mot de passe)
- Droits de connexion : voir `/etc/postgresql/9.4/main/pg_hba.conf` changer ligne IPv6 local connections puis reload

Eclipse

- Pour accéder à une BD, il faut un pilote : téléchargement ou
  `sudo apt-get install libpostgresql-jdbc-java`
- `Preferences / Data Management / Connectivity / Driver Definitions`
- Créer une instance de pilote pour PostgreSQL
- Connexions depuis vue *Data Source Explorer*
- Créer une connexion à BD (via pilote créé)
- Envoi de commandes : SQL Scrapbook (avec complétion)
- Édition de données : depuis Data Source Explorer
- Ouvrir vues ou éditeurs depuis Eclipse : voir `Window / Navigation / Quick Access`

Serveur d'application

- Avec un serveur d'application, il faut renseigner le pilote JDBC utilisé dans JNDI
- Conseil : s'assurer d'abord que l'instanciation fonctionne avec un projet simple Java SE et Maven
- Via l'interface d'administration, indiquer la source à utiliser pour la connexion JNDI par défaut
- Inclure le pilote JDBC PostgreSQL dans les bibliothèques du serveur d'application

Glassfish

- BD intégrée à Glassfish : Derby
- Il faut démarrer la BD : `asadmin start-database`
- Manuel SQL pour Derby
- Cependant pour ce cours on préférera PostgreSQL

Deux patterns principaux

Deux patterns courants liés à persistance : *Active Record* et *DAO*

- Active Record : chaque objet responsable de sa propre persistance
- Chaque objet contient méthodes CRUD : Create, Read, Update, Delete
- DAO ?

Deux patterns principaux

Deux patterns courants liés à persistance : *Active Record* et *DAO*

- Active Record : chaque objet responsable de sa propre persistance
- Chaque objet contient méthodes CRUD : Create, Read, Update, Delete
- DAO ? Data Access Object

Deux patterns principaux

Deux patterns courants liés à persistance : *Active Record* et *DAO*

- Active Record : chaque objet responsable de sa propre persistance
- Chaque objet contient méthodes CRUD : Create, Read, Update, Delete
- DAO ? Data Access Object
- Classes dédiées à interaction avec BD
- Permet isolation de cet aspect
- Typiquement utilisé avec Data Transfer Object

Références

- Patterns of Enterprise Application Architecture
- Core J2EE Patterns - Data Access Object

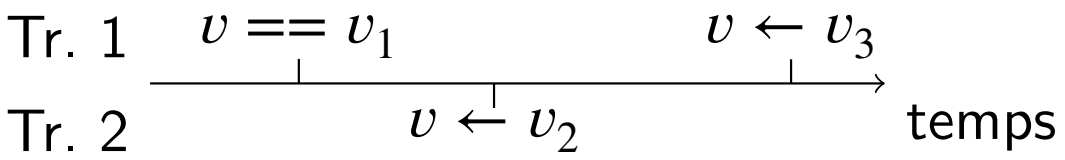
Nécessité des transactions atomiques

- Accès concurrents à DB : risques
- Si une transaction par statement ?

Nécessité des transactions atomiques

- Accès concurrents à DB : risques
- Si une transaction par statement ?

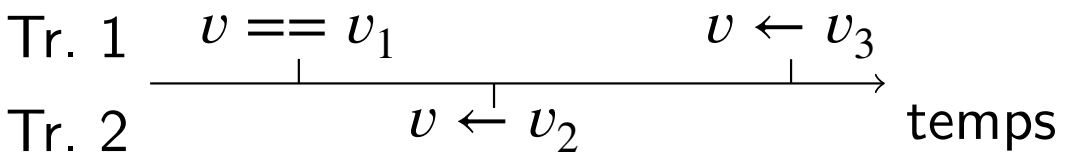
Lost update



Nécessité des transactions atomiques

- Accès concurrents à DB : risques
- Si une transaction par statement ?

Lost update

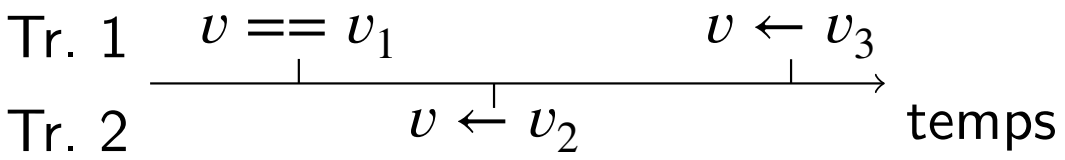


- Transaction atomique non triviale (couvrant un ensemble de statements) permet de lire-puis-écrire sans interruption
- Implémentation naïve : DB verrouillée pour un utilisateur pendant le temps de la transaction
- Problème ?

Nécessité des transactions atomiques

- Accès concurrents à DB : risques
- Si une transaction par statement ?

Lost update

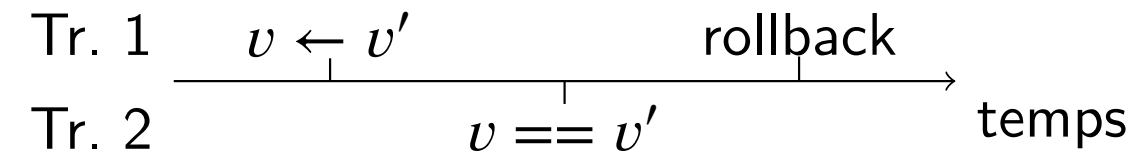


- Transaction atomique non triviale (couvrant un ensemble de statements) permet de lire-puis-écrire sans interruption
- Implémentation naïve : DB verrouillée pour un utilisateur pendant le temps de la transaction
- Problème ? Souvent trop peu efficace
- Protection : transaction terminée par *commit* ou *rollback*

Niveaux d'isolation

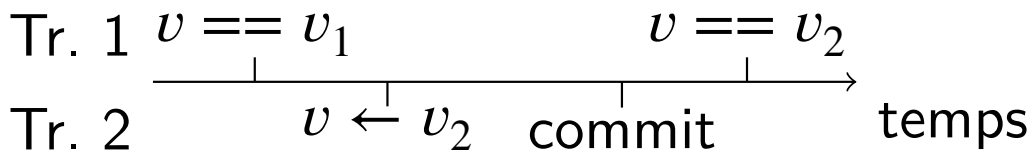
Read uncommitted (risques ↓)

Dirty read

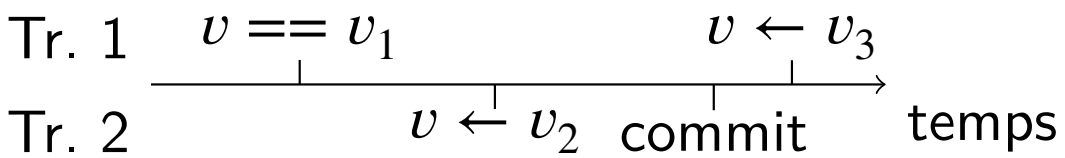


Read committed (protection ↑, risques ↓)

Non-repeatable rd

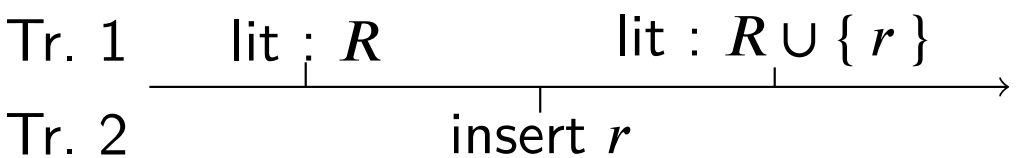


2^{nd} lost update



Repeatable read (protection ↑, risques ↓)

Phantom



Serializable (protection ↑)

Niveaux d'isolation

- Quatre niveaux d'isolation standards (ANSI ; JDBC ; JTA)
  (critiqués)
- Définis comme protection contre catégories de risques
- Risque défini comme : phénomène problématique
- SGBD configuré pour un niveau d'isolation donné
- Typiquement : Read committed
- Possible de se protéger contre certains risques au cas par cas

Protection contre 2nd lost update

- Optimiste : lire version lors lecture, check version lors écriture
- Pessimiste : verrouiller lors lecture

Exercices I

- Créer (sur papier) une table pour un type de votre projet.
- La créer dans votre BD via le SQL Scrapbook.
- Tester des requêtes simple de création, sélection, effacement dans le SQL Scrapbook.
- \+ [1] Permettre CR.D : Create, Read, Delete *aussi simple que possible*, via un ou plusieurs servlets ou GUI. (N'utilisez pas de paramètres complexes, ce n'est pas le but de cet exercice.)
- Programmer une méthode qui transforme un attribut d'un objet. Par exemple, elle met le nom en majuscule s'il ne l'était pas (obligation d'utiliser Java, pas SQL : supposez que la transformation est trop complexe pour être exprimée en SQL).

Exercices II

+ Permettre l'application de cette méthode (via un servlet ou via GUI). Votre servlet ne doit pas nécessairement accepter de paramètres. Attention à l'atomicité de la transaction !

1. Le + indique que cet aspect intervient dans la note

À faire

À faire *avant* le 16 mars

- Projet eclipse Java SE
- Dépendance Maven : H2
- Le prg affiche le numéro de version du pilote, puis quitte
- À faire par chaque membre de chaque groupe
- Envoyez au git de votre groupe dans rép. `user.name`
- Le prg logge « Démarrage » dans la console au démarrage, niveau INFO
- Configuration eclipse sur serveur git

Attention aux modifications suite à mise à jour

À faire (suite)

À rendre *avant* le 23 mars : par l'équipe

- Application accédant à une base PostgreSQL (Java SE + Maven ou Java EE)

- L'équipe conçoit des objets et tables pour les stocker
- Ces objets doivent avoir des connexions entre eux
- Votre conception devra soutenir les fonctionnalités futures de votre application
- Chaque membre choisit (au moins) un objet stocké dans la BD dont il aura la responsabilité principale

À faire (suite)

À rendre *avant* le 23 mars : par chacun

- Des servlets (Java EE) ou des fonctionnalités (Java SE)
- Qui permettent la création, lecture, effacement des objets dont vous êtes responsable ;
- Et qui permettent d'appliquer une transformation quelconque sur un attribut des objets en question (cf. Exercices)
- Partie de votre application de groupe

L'élégance de la conception et du code seront prépondérants dans l'évaluation

Auteurs

Chaque commit doit pouvoir être associé à ses auteurs

- Choisir un nom d'utilisateur une fois pour toutes et l'indiquer dans `user.name`
- Indiquer sur votre groupe MyCourse le lien entre identité réelle et `user.name` (si non évident)
- Je considère le vrai auteur comme celui indiqué par git
- Je tiens compte du reviewer éventuel

Pair programming apprécié

- Indiquez dans le commentaire du commit le binôme éventuel : « reviewer: Machin » avec Machin le `user.name`
- Mais chacun doit être auteur à part ~ égale

Licence