PDB

# Introduction

Le développement d’application Java avec une base de données relationnelle nécessite la conversion des données relationnelles en objets et vice-versa. Chaque objet devant être sauvegarder dans la base de donnée doit être converti et par conséquent, l’on devra dupliquer énormément de code similaire. La maintenance d’une telle application devient très difficile une fois que la base de données compte un nombre important de tables et de lignes. Pour pallier à ce problème, on utilise une technique de programmation qui permet de faire cette conversion de façon transparente, de telle sorte qu’on ait l’impression de travailler avec une base de données orientée objet.

Cette technique consiste à faire le lien entre les objets de l’application et la base de données. On parle de mapping objet-relationnel (en anglais object-relationnal mapping ou ORM). Cette correspondance entre objets et base de données se fait se fait par le biais de l’utilisation des frameworks (ensemble de composants logiciels). On peut siter : Apache ObJectRelationalBridge, Java Data Objects, Hibernate et bien d’autres. Dans notre cas, nous utilisons comme framework, Java Persistence API (API pour Applications Programming Interface).

Java Persistence API (JPA) est issue deJava Specification Requests (JSR), plus précisément la JSR 220. Le fait qu’elle persiste les données tout en créant et gérant la base de données de façon transparente aux développeurs est en fait son atout mais il peut également avoir un gros pris à payer. Mettons-nous dans le cadre d’un développeur, développant une application utilisant JPA pour la persistance avec une base de données décrite comme suite : une table A (ayant une cinquantaine de lignes et une dizaine de colonnes) en relation bidirectionnelle avec une table B (contenant des millions d’entrées et une dizaine de colonnes). Le développeur, en voulant lister les données de la table A où juste quelques colonnes lui seront intéressantes, pourra se retrouver avec comme résultat, la totalité de sa base de données. Ceci qui pourra fortement dégrader les performances de son application et même plus, si les données doivent transiter sur un réseau internet, une bonne partie de la bande passante sera utilisé inutilement.

L’utilisation de JPA ne devrait pas priver les développeurs de comprendre comment les relations seront faites, comment les tables seront modélisées en base de données et surtout ils devraient savoir qu’elle sera l’impact de leurs choix dans la base de données. D’où l’objet de notre étude. Elle se déroulera en 5 phases :

* Phase 1 : Prise en main status
* Phase 2 : Définir le modèle métier de test

Elle consiste à mettre en place un modèle de test de performance de notre application

* Phase 3 : Génération de données de test

Elle consiste à automatiser les données de test pour pouvoir faire des tests avec un très grand nombre de données.

* Phase 4 : instrumentation
* Phase 5 : Résultat

Elle sera réalisée sous la base des résultats des phases précédentes.

# Prise en main status

Elle consistera à comprendre le fonctionnement de JPA. Pour ce faire, on va développer une application ayant une interaction avec une base de données qui aura pour but, l’attribution et la suppression des cours à des étudiants et à des professeurs.

Cette application fera intervenir :

* Une base de données MySQL qui devra grandir et avoir une taille considérable lors de l’évolution du projet.
* JPA pour définir les objets métier (classes d’entités) qui serviront de lien avec la base de données.
* EJB (Staless Session Bean) qui jouera le rôle de serveur pour traiter les requêtes http et managera les classes qui serviront d’interface avec la base de données.
* Web services (JAX-RS) pour créer des services web avec une architecture REST.

On devra également créer un dépôt github nomé JPAPerf qui permettra de suivre l’évolution du projet et on utilise comme environnement de développement Netbeans 7.3, qui intègre le serveur Glassfish que nous utiliserons également. L’outil cURL nous permettra d’automatiser les requêtes (http).

## JPA et les classes d’entités

JPA repose essentiellement sur l’utilisation des annotations, introduites dans Java 5. Une classe d’entité est une véritable classe Java à laquelle on ajoute l’annotation « @Entity ». Cette annotation indique que la classe sera mappée dans la base de données et devra respecter les règles des Java Beans, c’est-à-dire :

* Posséder un constructeur sans argument. Ceci permettra d’instancier un objet de cette classe
* Posséder des accesseurs pour tous les attributs.
* Elle devra également comporter un attribut qui fera office de clef primaire dans la base données. Cette attribut est annotée par « @id » et sera de préférence un entier pour faciliter sa gestion par auto incrémentation.

