Université de Lille 1 SCIENCES ET TECHNOLOGIES

TP – Évaluation de l'algèbre relationnelle

Algorithmes des bases de données
Master Informatique – M1S2
2016–2017

1 Introduction

Dans ce TP nous allons considérer une implémentation très simpliste d'un SGBD, qui fait abstraction du problème de stockage des données sur disque. Par contre, la manière d'accéder à une relation est réaliste, c'est à dire, on accède aux relations en itérant leurs tuples.

Cette simplification vous permettra d'implémenter déjà les algorithmes de base d'évaluation d'expressions de l'algèbre relationnelle, sans vous préoccuper de l'accès disque.

L'implémentation est assurée par 4 classes, dont vous trouverez une description ci-dessous. Une description plus étoffée se trouve dans les commentaires javadoc des fichiers source.

Un tuple est représenté par un `String[]`. Ainsi, une relation est spécifiée par :

```
class SimpleDBRelation {
   SimpleDBRelation (RelationSchema schema);
   /** Permet d'itérer sur les tuples de la relation.
     * Retourne null lorsque l'itération est terminée. */
   public String[] nextTuple ();

   void switchToReadMode ();
   void switchToWriteMode();
   void addTuple (String[] tuple)
   RelationSchema getRelationSchema();
}
```

Une telle relation peut être utilisée en mode écriture ou en mode lecture. À la création elle est en mode écriture, ce qui permet d'ajouter des tuples (`addTuple`). Pour passer d'un mode à l'autre, on utilise les méthodes `switch...`. L'ajout de tuples est impossible en mode lecture, et `nextTuple` est impossible en mode écriture (ils lèveront une exception). Par ailleurs, `switchToReadMode` réinitialise l'itération.

Un schéma de relation spécifie le nom de la relation et sa sorte. Il propose également des méthodes utilitaires pour récupérer ou changer les valeurs des attributs d'un tuple de cette relation. Grâce à ces méthodes, vous pourrez systématiquement utiliser le nom d'un attribut (et pas sa position) pour accéder à sa valeur dans un tuple.

```
interface RelationSchema {
   public String getName();
   /** Les noms des attributs, considérés ordonnés. */
   public String[] getSort();

   /** La valeur dans tuple pour l'attribut donné. */
   String getAttributeValue(String[] tuple, String attributeName);
   /** Modifie la valeur dans tuple pour l'attribut donné. */
   void setAttributeValue(String newValue, String[] tuple, String attributeName);
   /** Crée un tuple vide (null partout) de la bonne taille. */
   String[] newEmptyTuple();
}
```

Nous fournissons une implémentation par défaut : `DefaultRelationSchema`.

Finalement, un SGBD est donné par

```
class SimpleSGBD {
   /** Crée une relation avec le nom et les attributs donnés en paramètre.
     * La relation ainsi créée est accessible par son nom. */
   SimpleDBRelation createRelation (String name, String[] sort);
   /** Retourne une relation précédemment créée, en utilisant son nom. */
   SimpleDBRelation getRelation (String name);
   /** Retourne un nom de relation qui n'existe pas encore
```

```
     * Utile pour faciliter la création de relations pour les résultats intermédiaires. */
   String getFreshRelationName ();
}
```

2 Algorithmes de base pour les opérateurs de la RA

Q 1.Télécharger sur Moodle l'archive `simpleSGBD.jar` qui contient les sources des 4 classes décrites ci-dessus.

Cette archive contient également une clases de test `TestTP2`. Celle-ci utilise les méthodes que vous devez implémenter pour ce TP.

Q 2. Créer la classe `TP2` qui est utilisée dans `TestTP2`, puis ajouter les en-têtes des méthodes nécessaires pour pouvoir compiler `TestTP2`

À propos de JUnit (introduction en 5 minutes, forcément incomplète). La bibliothèque JUnit permet d'écrire et exécuter des tests unitaires en java. Pour pouvoir l'utiliser, vous devez ajouter la librairie correspondante à votre classpath ; demandez à l'enseignante de l'aide si nécessaire. L'IDE vous permet ensuite de demander l'exécution des tests, ce qui produit un rapport de réussite :

- barre verte : le test passe
- barre bleu : le test ne passe pas car une assertion a été violée
- barre rouge : le test ne passe pas à cause d'une levée d'exception non prévue

Une assertion est une condition que l'on vérifie. Les assertions sont faites à l'aide de méthodes spéciales commençant par `assert`. Ce sont ces méthodes qui produisent le rapport de réussite.

Par exemple
`assertEquals(24, math.factorielle(4))`
vérifie que l'appel à la méthode `math.factorielle(4)` produit comme résultat 24. Le résultat de l'assertion est une barre verte, bleue ou rouge.

Tout au long du cours ABD nous allons vous fournir des classes de tests. Cela poursuit plusieurs objectifs :

- spécification : le test spécifie exactement ce qui est demandé. Si vous avez mal compris ce qu'une méthode doit faire et de ce fait l'avez mal implémentée, l'exécution du test vous le dira ;
- vérification : pour vérifier que votre implémentation répond à la spécification ;
- correction d'erreurs plus facile : pas besoin pour vous d'écrire un `main` pour tester le code. De plus, si on modifie le code et on introduit des erreurs dans les partie qui marchaient auparavant, l'exécution des tests pourrait les détecter ;

Les tests que nous fournissons serviront aussi pour noter votre projet, en identifiant les fonctionnalités supportées. Ainsi, exécuter régulièrement les tests que nous vous fournissons est indispensable. Nous vous encourageons **très fortement** d'écrire aussi vos propres tests.

Q 3. Exécutez la classe `TestTP2`. Clic droit sur le nom de la classe, puis `Run as ...` → `JUnit test` dans Eclipse. Les autres IDE proposent des mécanismes similaires.

Vous voyez des barres bleues ou rouges. C'est normal, vous n'avez pas encore implémenté les méthodes utilisées dans les tests.

Q 4. Implémenter les méthodes de la classe `TP2` jusque faire passer tous les tests au vert.