Université de Lille 1 SCIENCES ET TECHNOLOGIES

TP – Évaluation de l'algèbre relationnelle (suite)

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

Vous devez finir d'implémenter la classe `TP2` avant de commencer ce TP.

Les objectifs de ce TP sont :
- se familiariser avec la hiérarchie de classes `RAQuery` pour représenter les requêtes ;
- commencer l'écriture de code qui permet d'évaluer des requêtes de type `RAQuery`.

Préparation Récupérer l'archive `tp3.zip` sur Moodle qui contient :
- la classe `TP3` et sa classe de test `TestTP3` ;
- le package contenant la hiérarchie de classes `RAQuery` ;
- l'interface `PhysicalOperator` ;
- la classe `VolatileRelationSchema` qui permet de créer des schémas de relations anonymes.

Sauvegardez la classe `TP2` (au cas où), puis décompressez l'archive `tp3.zip` dans le répertoire de sources de votre projet.

La classe `TP3` que vous devez écrire contient l'unique méthode

```
PhysicalOperator getOperator(RAQuery query, SimpleSGBD sgbd);
```

Cette méthode prend en entrée une requête et retourne un objet qui permet d'itérer sur le résultat de la requête.

L'écriture de cette méthode nécessite l'écriture de plusieurs classes.

Nous vous suggérons deux manières différentes pour implémenter cette méthode, décrites dans les Section 1 et Section 2 respectivement. Une version qui mixe les deux est suggérée dans la Section 3. Vous devez choisir une de ces manières de procéder, après avoir lu les avantages et inconvénients de chacune.

1 Avec relations volatiles

Il s'agit d'implémenter la hiérarchie de classes `PhysicalOperator`. Dans ce cas l'objet retourné par `getOPerator` est un arbre similaire à celui de la requête, mais dont les noeuds sont de type `PhysicalOperator` et non de type `RAQuery`.

Les avantages de cette méthode sont :
- fidèle à la manière d'implémenter un système de gestion de bases de données ;
- permet d'intégrer toutes sortes d'optimisations pour l'évaluation de requêtes ;
- la seule à pouvoir garantir une évaluation efficace des requêtes.

L'inconvénient est :
- plus difficile à aborder, donc plus grand risque d'échec.

Si vous choisissez cette manière de faire, vous devez commencer par écrire au moins cinq classes qui implémentent `PhysicalOperator`, une pour chacun des opérations sélection, projection, renommage, jointure et "relation stockée". Cette dernière permet simplement d'itérer sur les tuples d'une relation stockée, et sera utilisée comme feuille dans un arbre de type `PhysicalOperator`.

Ensuite vous devez écrire un algorithme qui à partir d'un objet de type `RAQuery` construit un objet de structure similaire mais de type `PhysicalOperator`.

Voici les étapes recommandées :

Q 1. Écrire une classe `SequentialAccessOnARelationOperator implements PhysicalOperator` qui prend à la construction un `SimpleSGBD` et un nom de relation, et qui retourne un itérateur sur les tuples de cette relation. Notez que la classe `SimpleDBRelation` fournit déjà un itérateur. Il suffira de déléguer la méthode `nextTuple` à cette classe, et écrire uniquement les deux méthodes restantes de `PhysicalOperator`.

Q 2. Écrire la classe `ProjectionOperator implements PhysicalOperator` qui prend à la construction les noms des attributs pour la projection, ainsi qu'un opérateur qui servira d'entrée. La méthode `nextTuple` est la plus facile à écrire pour une projection car on retournera tous les tuples de l'itérateur sous-jacent.

Q 3. Écrire juste ce qu'il faut dans la méthode `getOperator` de `TP3` pour passer les tests portant sur la projection. C'est à dire, vous allez tester si le paramètre est une projection d'une table stockée et lever une exception si non. Si le paramètre est bien une projection d'une table stockée, alors vous devez créer le `PhysicalOperator` qui correspond et le retourner. Il sera construit en utilisant `ProjectionOperator` et `SequentialAccessOnARelationOperator` seulement.

Q 4. Écrire la classe `SelectionOperator implements PhysicalOperator` qui prend à la construction un opérateur en entrée, ainsi que le nom d'un attribut, une valeur constante et un opérateur de comparaison de type `ComparisonOperator` qui définissent la sélection. Cet itérateur est un peu plus difficile que la projection, car `nextTuple` devra potentiellement avancer plusieurs fois de suite dans la relation sous-jacente avant de retourner un tuple du résultat.

Q 5. Ajouter juste ce qu'il faut dans la méthode `getOperator` de `TP3` pour passer les tests portant sur la sélection. Suivez les conseils donnés pour la projection.

Q 6. Écrire la classe `JoinOperator implements PhysicalOperator` qui prend à la construction deux opérateurs `left` et `right`. C'est l'itérateur le plus difficile à écrire. Une manière de simplifier est d'implémenter un opérateur de produit cartésien, qui retournera la suite de couples constitués d'un tuple de chacun des opérateurs `left` et `right`. Vous pouvez ensuite faire le filtrage sur ces tuples de manière similaire à la sélection. N'oubliez pas que dans `TP2` vous avez déjà implémenté tout ce qu'il faut pour tester si deux tuples appartiennent à la jointure et pour les joindre.

Q 7. Ajouter juste ce qu'il faut dans la méthode `getOperator` de `TP3` pour passer les tests portant sur la jointure. Suivez les conseils donnés pour la projection.

Si vous avez réussi les tests des questions précédentes, vous pouvez avoir une relative confiance en le bon fonctionnement de vos itérateurs.

La prochaine étape consistera d'enlever les restrictions mises dans la méthode `getOperator` de `TP3` pour qu'elle puisse construire le bon opérateur pour chaque requête. Ce sera le sujet d'un prochain TP.

2 Sans relations volatiles

Il s'agit d'utiliser les fonctions `computeSelection`, `computeProjection` et `computeJoin` écrites la semaine dernière. Dans ce cas, le résultat de l'évaluation de la requête est une relation écrite sur disque et l'objet retourné par la méthode `getOperator` de `TP3` est un simple itérateur sur cette relation, qui est déjà fourni par la classe `SimpleDBRelation`.

Les avantages de cette méthode sont :

- ne nécessite pas l'écriture des itérateurs, ce qui la rend plus facile à réaliser. En effet, le calcul d'une jointure par itération n'est pas trivial ;
- nécessite un moindre niveau d'abstraction, ce qui la rend sans doute plus facile à comprendre,appréhender et implémenter.

L'inconvénient est :

- ne permet pas d'avoir une évaluation efficace des requêtes.

Ainsi, cette manière de procéder est à choisir uniquement si vous avez pris du retard avec `TP2` et vous craignez de ne pas réussir à implémenter les itérateurs pour les relations volatiles.

Voici quelques indications pour la marche à suivre. Discutez avec l'enseignante pour plus d'explications.

Vous devez écrire la classe `NonVolatileOperator implements PhysicalOperator`. Cette classe permet de parcourir l'arbre de type `RAQuery` des feuilles vers la racine. Sur chaque noeud de l'arbre il faut faire appel à l'une des fonctions du `TP2` (`copmuteSelection`, `computeProjection` ou `copmuteJoin`) pour évaluer l'opération spécifiée par ce noeud. Il faut par ailleurs pouvoir accéder aux noms des relations qui contiennent les résultats intermédiaires déjà calculés. La solution la plus simple consisterait à utiliser une table de hachage (`Map` en java) qui à chaque noeud de l'arbre de type `RAQuery` associe le nom de la relation résultat calculée dans ce noeud.

3 Méthode mixte

Il s'agit d'implémenter les itérateurs comme expliqué dans la Section 1, sauf pour la jointure dont l'itération est plus difficile à faire.

Cette manière de faire est à choisir seulement si vous avez commencé avec les relations volatiles mais vous n'avez pas réussi à implémenter la jointure, et vous avez pris du retard.

Dans ce cas chaque bloc de sélections et projections regroupés dans l'arbre sera évalué par un itérateur, tandis que le résultat d'une jointure sera évalué par la méthode `computeJoin` du `TP2` et donc systématiquement écrit sur disque.

Dans tous les cas discutez avec l'enseignante si vous prenez cette option.