Université de Lille 1 SCIENCES ET TECHNOLOGIES

TP6 – Gestion de la mémoire

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

L'objectif du TP est de comprendre comment gérer des relations stockées sur un disque et en ayant une mémoire finie. Nous verrons comment cette mémoire finie limitante pour l'implémenation des opérateurs physique

Pour la suite de projet, nous supposerons que les tuples ne sont accessibles qu'en chargeant les pages sur lesquels ils sont stockés. Une page n'est accessible que par son adresse qui est résolue par le MemoryManager. Ce MemoryManager ne peut stockée en mémoire vive qu'un nombre limité de pages.

Pour simplifier la manipulation, nous supposerons que les pages sont une représentation abstraite d'une page réelle contenant une liste de Tuples ainsi que les adresses des pages précédentes et suivantes auxquelles elle est liée. Elle possède également sa propre adresse.

Nous commencons par implémenter une nouvelle notion de stockage pour les relations ent utilisant la mémoire. Ensuite nous revisiterons par les opérateurs physique de tables, la sélection et la projection et du join Toutes les classes à écrire lors de ce TP seront dans le package `univlille.m1info.abd.phys`.

Attention, pour obliger l'utilisation du MemoryManager, il est interdit d'utiliser trois variables stockant des pages dans vos implémentations.

Avant de passer à ce TP, finir les Tp précédants

1 Page

L'interface Page permet de stocker un nombre limité de tuples. Pour commencer la lecture des tuples, il faut utiliser la méthode public `switchToReadMode()` qui initialise un itérateur. Le tuple suivant est accessible par la méthode `nextTuple()`.

2 Memory Manager

La mémoire vive et les accès disques sont simulés au travers une classe ManagerMemory : Cette classe possède un nombre finie de pages qu'elle peut stockée en mémoire vive. Les autres pages sont stockées dans un disque virtuel (simulé dans le cadre de ce TP).

- Il est possible de demander à avoir accès à une page en mémoire en utilisant la méthode `loadPage(int pageAddress)` qui retourne la page associée. Si cette page était stokée en mémoire vive alors elle utilise cette valeur sinon elle doit chargée la page du disque vers la mémoire vive.
- Il est possible de mettre à jour une page en utilisant la méthode
  `PutinMemory(Page Datapage, int Address)`
- Il est possible de créer une nouvelle page associée à une nouvelle adresse sur le disque par `NewPage(int Arity)`. Comme les pages stockent des tuples d'une arité précise, il faut préciser cela en entrée. La page retournée est vide.
- Il est possible de libérer un slot de la mémoire vive en utilisant la méthode
  `releasePage(int pageAddress, boolean persistant)`

2.1 Calcul des opérations sur disques

Différentes méthodes de Memory Manager permettent de donner les différents couts que des exécutions des opérations que vous avez implémenté ont obtenus. Les scores de votre implémentation seront basés sur ces couts.

2.2 Instentiation

Dans le fichier defaut.jar, vous trouverez deux instantiations des interfaces `Page` et `MemoryManager`, appelés respectivement `Defaultpage` et `SimpleMemoryManager`. Voici les deux façons de créer des objets pour ces classes :

```
public SimpleMemoryManager (int memorySize, int pageSize) throws IOException
```

`memorySize` indique le nombre de pages qui peuvent être stockées en mémoire vive et `pageSize` indique la contenance d'une page.

```
public Defaultpage (int  Add, int Arity, int Capacity)
```

`Add` est l'adresse de la page. `Arity` est l'arity des tuples stockés dans cette page et `Capacity` et le nombre maximal de tuples qui peuvent être stockés dans la page.

3 Représentation d'une relation stockée sur le disque

Nous utilisons la classe `DefaultRelation` qui implémente une relation stockée sur disque.

Q 1. Implémenter la méthode `loadTuples` qui met en mémoire une liste de tuples. L'ajout de ces tuples redéfinie l'instance de la relation, ils ne sont pas ajoutés.

4 Opérateur physique

Rappelons que pour le moment opérateur physique produit un ensemble de tuples en sortie, et consomme éventuellement des tuples en entrée. Les opérateurs sont implémentés comme des itérateurs. Un opérateur physique doit satisfaire cette interface. Pour respecter que maintenant, nous avons ajoutons une nouvelle méthode pour les opérateurs physique :

```
public int nextPage();
```

Vous devez rajouter cette ligne dans l'interface opérateurs physique . Cette méthode doit retourner l'adresse de la prochaine Page de tuples calculés par cet opérateur. Ainsi un opérateur ne consomme plus directement de tuples mais des pages qui sont données par les adresses physiques car nous ne savons pas apriori si cette page est toujours en mémoire vive.

5 Opérateurs sur les tables

Les opérateurs de base produisent comme sortie des pages qui sont issus directement d'une table. On cherchera seulement à parcourir la relation de façon séquentielle.

Rappelons que pour les TPs précédents, nous avions convenu que chaque table a son propre répertoire dédié où sont stockées les données et méta-données. Dans ce cadre nous supposons qu'une table est représentée par une instance de la classe `DefaultRelation`.

5.1 Opérateur de parcours séquentiel de table

Nous souhaitons implémenter un opérateur physique pour parcourir une relation representée par une instance de la classe `DefaultRelation`. Un tel opérateur doit énumérer toute les lors des appels successifs à `nextPage()`.

Q 1. Écrire une classe qui implémente l'interface `PhysicalOperator` et permet un parcours séquentiel de tous les tuples d'une table.

6 Opérateurs de filtrage

Nous rappelons que l'opérateur de filtrage consomme des tuples produits par un autre opérateur et leur applique un filtre. C'est une généralisation de la sélection et de la projection. Dans notre cadre, nous revisitons l'implémenation d'un filtre dans le cadre où le filtre lit des pages en entrées et non des tuples. La difficulté principale est qu'un filtre ne retourne une page que lorsque celle-ci est pleine ou les pages de l'opérateur précédent ont toute été lues.

Q 1. Écrire la méthode `nextPage()` pour votre classe de filtre et permet un parcours séquentiel de tous les tuples d'une table.

7 Opérateur Join

Nous nous proposons de regarder comment un join est implémenter.

Q 1. Implémenter dans le cadre classique du TP3 de l'opérateur Join qui implémente l'interface `PhysicalOperator`.

Q 2. Implémenter la méthode `nextPage()` pour votre classe Join.

8 Triées une relation

Le but est de triées une relation.

8.1 Algorithme de trie

Q 1. En utilisant un algorithme de trie de votre choix, trié une table stocké sur disque accessible au travers MemoryManager. Par simpliciter nous supposerons que les pages sont contigues sur le disque.

8.2 Algorithme de Join

Q 2. En utilisant l'algorithme de SortJoin, définir une classe qui implémente le Join et qui est correct quand les relations en entrées sont triées.