TP 2025 Fil Rouge « Je code mon SGBD »

4ASTI – INSA CVL – B. Nguyen & P. Clemente

Objectifs généraux du projet

L'objectif du projet est en mettre en application vos connaissances sur les mécanismes internes des SGBD par l'implémentation d'un certain nombre d'opérateurs et de techniques présentées dans le cours, en particulier la gestion des index. L'idée est d'aller aussi loin dans le TP que possible.

Partie I- Recoder les blocs sous forme de fichier

Operateur DisqueBloc/FullScanDisqueBloc

Un code un peu complexe est proposé dans la méthode FullScanTableDisque pour représenter les blocs. Dans ce TP, on va écrire une nouvelle classe (DisqueBloc) qui va utiliser une représentation d'un fichier sous forme de blocs sur le disque de la manière suivante : on va en fait créer 1 fichier par bloc, comme ça on peut voir comment le fichier va être construit. On se donne un nom de fichier (ex. table1) on va en fait créer plusieurs fichiers : table1.bloc1, talble1.bloc2 etc. On décide à l'avance (par exemple via une constante) le nombre d'enregistrements que peut contenir un bloc. La classe DisqueBloc devra permettre de générer une table contenant des données aléatoires et de la stocker sur le disque dans ce format.

Une des objectifs est de compter combien de blocs on lit, ce qui peut se faire assez facilement en mettant un compteur à chaque fois qu'on fait un appel à la méthode de lecture. L'entête de chaque fichier contiendra 3 entiers (voir l'exemple de FullScanTableDisque pour la gestion des fichiers et de l'entête) : int : nombre de colonnes de la table, int : nombre de tuples dans le bloc, int : numéro du bloc suivant (ou 0 si c'est le dernier bloc)

Montrez que votre code fonctionne en créant un fichier composé de 3 blocs : table1.bloc1, table1.bloc2, table1.bloc3 avec une table de 4 colonnes et 10 lignes, sachant qu'on peut stocker 4 tuples dans chaque bloc, puis en affichant son contenu via l'operateur FullScanDisqueBloc (d'interface Operateur) que vous aurez aussi écrite.

Partie II- Construction d'un index sur DisqueBloc

Index par hachage statique

En utilisant une partie de la fonction de hachage MD5 (par exemple le premier octet), construisez une classe IndexHachageStatique qui permet de dire dans quel bloc se trouve une donnée, en fonction de sa valeur par la fonction de hachage statique. Codez aussi une méthode qui charge un fichier (sous forme de DisqueBloc, par exemple table1.bloc1, table2.bloc2 etc) et le réécrit en utilisant l'index avec le nom table1.index.bloc1, table1.index.bloc2 etc).

Index par hachage dynamique

Faire la même chose, mais avec un index dynamique.

Opérateur de jointure sur index

Opérateur de jointure sur index

Codez une classe JointureSurIndex (d'interface Operateur) qui implémente la jointure entre deux tables, dont au moins une est indexée sur l'attribut de jointure. L'algorithme est le suivant : on parcourt l'ensemble des tuples de la table non-indexée, et pour chaque tuple, on fait un appel à la table indexée pour trouver les tuples à joindre. Montrez par une requête que votre code fonctionne correctement.

Opérateur de jointure par tri-fusion

Codez une classe JointureTriFusion (d'interface Operateur) qui code cet algorithme de jointure. On fait l'hypothèse ici qu'on dispose d'assez de mémoire pour faire le tri en mémoire.

Partie III- Optimiseur

Optimiseur heuristique simple

Construire un mini optimiseur qui prendrait un arbre d'exécution et le réorganiserait en poussant les restrictions vers le bas, rajouterait les projections, et changerait les double boucles pour faire des boucles sur index, si l'index existe. Montrez via un programme main si votre optimiseur est utile (par exemple en comptant le nombre de blocs lus sur le disque).