Projet bases de données – L3 Info – 2025

Guillaume Scerri – guillaume.scerri@lmf.cnrs.fr,
Théo Vignon – theo.vignon@lmf.cnrs.fr

1 Description générale

Le but de ce projet est de réaliser un système de gestion de bases de données (SGBD dans la suite), capable d'éxécuter des requêtes SQL, y compris des requêtes de création/modification de données, sur des données stockées sur disque. Le projet sera fait par groupe, chaque groupe réalisant une partie du système et devant s'interfacer avec les autres groupes. Le projet sera évalué sur trois rendus pour chacun des groupes :

- l'interface des modules à rendre pour les différents groupes (à discuter entre les groupes), **avec leur documentation**,
- une implémentation naïve (i.e. sans optimisation) des interfaces et la documentation correspondante, il s'agit principalement ici de faire un test d'intégration des différentes parties,
- une implémentation au moins partiellement optimisée, avec la documentation des optimisations réalisées.

Il sera demandé à chaque groupe d'implémenter un module. On décompose ici l'architecture d'un SGBD en quatre modules.

- Un compilateur de SQL, qui fournit en sortie un *arbre algébrique abstrait*.
- Un générateur de plan d'éxécution prenant un arbre algébrique abstrait en entrée et construisant un *plan d'éxécution physique* optimisé pour le modèle de coût des algorithmes de gestion de données.
- Des *algorithmes de gestion de données*, utilisant le modèle physique de données et les indexes éventuels pour l'optimisation, et proposant un *modèle de coût* dépendant des statistiques des tables et de la présence d'indexes.
- Un *module de stockage de données* maintenant le modèle de données physique des tables, et des indexes, et exposant des méthodes d'accès aux données, ainsi que des statistiques sur les différentes tables.

Le projet est divisé en trois volets pour organiser la production de code et s'assurer d'une certaine synchronisation entre les groupes. On attend pour l'ensemble des rendus un bonne documentation du code, idéalement avec Ocamldoc.

- D'abord on construira des interfaces entre le code des différents groupes, avec un attention particulière au fait que ces interfaces permettent un certaine indépendance entre les groupes.
- Ensuite on proposera une implémentation naïve mais fonctionnelle, le but ici est que tous les groupes puissent se reposer sur une implémentation minimale des autres groupes dans la partie finale. On développera dans la séance suivante un jeu de tests des différentes parties et de l'architecture entière et on mettra en place de l'intégration continue sur git.

— La partie finale consiste en l'implémentation de primitives plus complexes et d'optimisations.

La notation sera 1/3 rendus intermédiaires (principalement l'implémentation naïve) et 2/3 rendu final et soutenances.

2 Dates importantes

Constitution des groupes Vendredi 31 janvier

Rendu des interfaces Mercredi 12 février 12h

Rendu implémentation naïve Mercredi 26 mars 12h

Rendu final Mercredi 21 mai 12h

Soutenances Vendredi 23 mai

3 Détail des parties

3.1 Compilation SQL

On s'intéresse dans cette partie à créer un compilateur d'un fragment de SQL vers des arbres de requêtes algébriques. On divise en fait le langage en deux parties, le langage de requêtes (qui ne modifie pas les données sous jacentes, i.e. les requêtes commençant par `Select`) et le langage de manipulation de données qui créée des tables, et insère, supprime et modifie leur contenu.

3.1.1 Langage de requêtes

Pour le langage de requêtes le compilateur devra vérifier des conditions avant de produire un arbre algébrique.

1. Les requêtes sont bien formées syntaxiquement.
2. Les tables correspondantes existent bien (interface avec le groupe stockage de données).
3. Les types des différents attributs sont compatibles avec la requête (interface avec le groupe stockage de données).

Une fois la requête SQL analysée, le compilateur devra produire un arbre algébrique abstrait (interface avec le groupe optimisation).

Attendus

Rendu 1 Au moins le type arbre algébrique abstrait.

Rendu 2 Compilation du fragment sans requêtes imbriquées, avec des conditions étant uniquement des conjonctions/disjonctions d'égalités et sans agrégats.

Rendu final Compilation de SQL avec des restrictions à discuter. Par exemple : agrégats, requêtes imbriquées, `null`, ...

3.1.2 Langage de manipulation

Pour le langage de manipulation de données, il s'agit d'implémenter `Create table`, `Insert`, `Delete`, `Update`. Pour la création de table on devra aussi stocker les contraintes de clés étrangères, et donner les indexes primaires et secondaires à créer ainsi que possiblement d'autres contraintes à terme. La charge de vérifier que les autres requêtes de manipulation satisfont ces contraintes sera à la charge du compilateur.

Attendus

Rendu 1 L'interface de création de tables, d'indexes et le format des contraintes à stocker par le groupe stockage de données.

Rendu 2 Création de tables avec des indexes sur les clés primaires, insertion de données.

Rendu final Mise à jour des données et suppression, ainsi que vérification des contraintes. Indexes secondaires.

4 Optimisation

La partie optimisation est responsable de produire un plan d'éxécution physique à partir d'un arbre algébrique. Ce groupe devra aussi optimiser le coût d'éxécution des requêtes en fonction d'un modèle de coût des différents algorithmes proposé par le groupe algorithmique des données.

Attendus

Rendu 1 Interface plan d'éxécution physique avec le groupe algorithmique des données.

Rendu 2 Traduction arbre algébrique vers plan d'éxécution pour les opérateurs de base de l'algèbre relationnelle (jointure, restriction, projection, union, intersection, différence, produit cartésien, renommage), et calcul du coût.

Rendu final Optimisation du coût, traitement des agrégats, possiblement traitement intelligent de certaines conditions et des requêtes imbriquées.

5 Algorithmique des données

Le but de cette partie est de proposer des algorithmes implémentant les opérateurs de l'algèbre relationnelle, ainsi qu'un évaluation des coûts de calcul de ces opérateurs.

Attendus

Rendu 1 Interface plan d'éxécution physique avec le groupe Optimisation, et interface d'accès aux données avec le groupe stockage.

Rendu 2 Implémentation naïve des opérateurs de jointure, projection, restriction, renommage, union, intersection et différence.

Rendu final Implémentation d'algorithmes efficaces, utilisant les indexes intelligemment (par exemple sort-merge join et/ou grace hash join). Implémentation des agrégats (group by, having).

6 Stockage de données

Le but de cette partie est d'implémenter le stockage physique des données sur disque, ainsi que des méthodes d'accès efficaces à travers des indexes.

Attendus

Rendu 1 Interface d'accès aux données, exposée au groupe algorithmique. Interface d'insertion de données, exposée au groupe compilation. Interface de présentation de la structure (y compris contraintes) et statistiques exposée à tous les autres groupes.

Rendu 2 Implémentation naïve d'un stockage, par exemple dans un fichier CSV, trié par clé primaire (avec insertion inefficace).

Rendu final Implémentation efficace, avec insertion et accès efficace, en utilisant des arbres B+, ou des indexes par hachage. Implémentation d'indexes secondaires.