Projet de « Bases de Données I »
Dépendances fonctionnelles et normalisation

Pierre Hauweele

2019 – 2020

1 Description du projet

1.1 Introduction

L'objectif de ce projet est d'implémenter un outil de gestion de dépendances fonctionnelles (DF) sur des bases de données. Cet outil, écrit dans le langage de votre choix [1], doit se présenter sous la forme d'une librairie/module (propre au langage), ainsi que d'un utilitaire en ligne de commande.

Cet outil doit permettre d'appliquer des opérations sur des dépendances fonctionnelles (restriction sur un schéma, clôture d'un ensemble d'attributs [2], vérification d'une conséquence logique), de déterminer les clés et superclés [3] d'un schéma, de vérifier qu'un schéma est en BCNF ou 3NF et d'en proposer des décompositions le cas échéant, ainsi que de vérifier qu'une base de données (SQLite) satisfait un ensemble de DF.

1.2 Comportement et fonctionnalités de l'outil

La plupart des SGBD permettent de définir un ensemble de contraintes d'intégrité telles que les clés primaires ou les clés étrangères. Les dépendances fonctionnelles sont rarement supportées par ces SGBD. Dans le cadre de ce projet, vous devez utiliser SQLite qui ne propose pas ce type de contraintes d'intégrité.

Afin de pouvoir gérer les DF, nous supposerons que toute base de données contient une relation nommée `FuncDep` qui contient des triplets de chaînes de caractères de la forme `(table_name, lhs, rhs)`. Un tel triplet encode une DF singulière `lhs` $\rightarrow$ `rhs` pour la relation désignée par le nom `table_name`. La chaîne `lhs` représente une liste d'attributs séparés par un espace. Par exemple, le triplet `("Employee", "name dept", "salary")` définit sur la relation `Employee` la DF `name`, `dept` $\rightarrow$ `salary`. Notez que puisque les DF sont singulières, `rhs` ne contient qu'un seul attribut.

Si la base de données ne contient pas de relation `FuncDep`, vous supposerez que l'ensemble des DF est vide et l'ajouterez si nécessaire.

1. C, C++, Java, OCaml, Python. Pour un autre langage, demander à l'assistant.
2. L'ensemble des attributs impliqués par des DF à partir de l'ensemble d'attributs de départ.
3. Une superclé est un sur-ensemble d'une clé.

1.2.1 Lecture et écriture des dépendances fonctionnelles

L'application doit permettre de lire la relation `FuncDep` afin d'identifier les différentes dépendances fonctionnelles associées à la base de données. Il doit être possible d'ajouter facilement une dépendance fonctionnelle, de supprimer ou de modifier une dépendance fonctionnelle existante.

L'utilitaire (en ligne de commande ou via une interface grapique), doit fournir une interface interactive pour définir les dépendances fonctionnelles. Il est attendu que l'utilisateur n'ait pas à mémoriser ou à utiliser une syntaxe complexe pour ajouter des dépendances fonctionnelles.

L'application peut éventuellement proposer des fonctions permettant de suggérer automatiquement des dépendances fonctionnelles en tenant compte du contenu de chaque relation. Sur base de ces suggestions, l'utilisateur pourrait alors sélectionner les dépendances fonctionnelles qu'il considère comme étant adéquates.

1.2.2 Vérification des dépendances fonctionnelles

L'application doit permettre l'affichage des différentes dépendances fonctionnelles pour chaque relation. Il est attendu que l'application puisse effectuer des opérations élémentaires sur les dépendances fonctionnelles, notamment :

1. afficher les dépendances fonctionnelles qui ne sont pas satisfaites. Pour ces dépendances, vous pouvez éventuellement proposer l'affichage des tuples concernés et éventuellement la suppression de certains d'entre eux ;
2. distinguer les dépendances fonctionnelles qui sont une conséquence logique d'autres dépendances fonctionnelles et éventuellement permettre à l'utilisateur de voir ces dernières ;
3. supprimer sélectivement, et interactivement, les dépendances fonctionnelles inutiles ou incohérentes : les conséquences logiques, les dépendances non satisfaites, les dépendances concernant des relations ou des attributs n'existant pas (par exemple, concernant des relations ou attributs supprimés via une autre application), etc.

1.2.3 Identification des clés et des superclés

L'application doit être capable de lister, à l'aide des dépendances fonctionnelles, les clés et superclés pour chaque relation. Notez bien qu'il s'agit de calculer des clés et des superclés sur base des dépendances fonctionnelles, et non pas de lire ou modifier le schéma des tables SQLite pour ajouter ou lister des clés primaires qui seraient définies via SQLite.

1.2.4 Vérification des normes BCNF et 3NF

L'application doit permettre de vérifier si la base de données respecte les normes BCNF et 3NF. Pour chaque norme, il doit être possible d'identifier les relations qui ne respectent pas la norme et d'identifier les dépendances fonctionnelles qui sont concernées. Le fait qu'une base de données soit en BCNF ou 3NF ne dépend que des dépendances fonctionnelles et non pas de son contenu.

Si la base de données ne respecte pas 3NF, et sous la condition que la base de données respecte les différentes dépendances fonctionnelles, il doit être possible

d'exporter cette dernière dans une nouvelle base de données au format 3NF. Pour cela, vous devez calculer une décomposition 3NF qui préserve le contenu et les dépendances fonctionnelles. Bien entendu, la nouvelle base de données doit contenir une relation `FuncDep` avec les données adéquates.

2 Organisation

2.1 Langage et gestionnaire de versions

Vous pouvez réaliser ce projet dans le langage que vous souhaitez parmi le C, C++, Java, OCaml, Python. Chacun de ces langages possède une librairie *sqlite3* permettant d'interfacer des bases de données SQLite. Vous pouvez utiliser leurs librairies standard ainsi que les librairies classiquement utilisées dans ces langages (e.g. *Boost* pour C++, *Core* pour OCaml). L'utilisation d'une librairie implémentant déjà les fonctionnalités demandées est bien sûr proscrite.

Votre projet doit également être accompagné d'un moyen facile de compilation (e.g. *Ant* pour Java [4], *Makefile* pour C/C++, *Oasis* pour OCaml). Ainsi que d'un fichier `README` indiquant comment il faut compiler/lancer votre application.

Dans le cadre de ce projet, il vous est demandé d'utiliser activement un gestionnaire de versions (Git ou Bazaar). Un outil de contrôle de version vous assiste dans la réalisation et l'implémentation de projets en gardant une trace des modifications effectuées dans le temps. Ce type d'outils facilite le suivi des modifications, le travail collaboratif et le développement de fonctionnalités en parallèle. Git et Bazaar sont des outils de contrôle de versions dit décentralisé au sens où ils ne nécessitent pas d'un serveur central pour fonctionner : ils fonctionnent en effet uniquement localement et les synchronisations éventuelles sont effectuées par des protocoles classiques (HTTP, SSH, ...) par opposition à d'autres outils comme CVS ou SVN.

Utiliser un outil de contrôle de versions est une bonne habitude à prendre dans le cadre de travaux de groupe. Pour les projets individuels, cela permet de suivre également plus facilement son travail, et la surcharge induite par l'utilisation d'un tel outil est vraiment faible. Pour plus d'informations sur Bazaar, vous êtes invités à consulter le site officiel : `http://bazaar.canonical.com`. Pour Git, consultez `https://git-scm.com`. Vous pouvez utiliser des service de synchronisatio nde dépot en ligne comme `https://bitbucket.org` ou en encore `https://github.com`.

2.2 Remise du travail

Il s'agit d'un projet à réaliser par groupe de deux étudiants, dans lequel il vous est imposé d'utiliser l'outil de gestion Bazaar (ou git) pour organiser votre travail. Veuillez envoyer la composition de vos groupes, ainsi que votre choix de langage par e-mail à l'adresse pierre.hauweele@umons.ac.be pour le **jeudi 21 novembre**.

Le travail doit être livré sous la forme d'une archive (.zip ou un format libre (tar, gz, xz, etc.) portant, en majuscules uniquement, les noms et prénoms des étudiants séparés par un tiret. L'archive doit contenir votre application, le fichier `README`, votre système de compilation et le répertoire `.bzr` (ou `.git` si vous

4. cfr. le projet d'informatique de première.

utilisez git). Si vous souhaitez fournir d'autres éléments (documentation auto générée, fichiers tests, remarques générales, etc.), placez les dans un répertoire nommé `misc`. N'oubliez pas de synchroniser votre travail via Bazaar/Git et de valider vos derniers changements avant de remettre l'archive.

L'archive est à déposer sur Moodle pour le **lundi 23 décembre à 23h55** au plus tard.

Vous serez évalué sur les qualités fonctionnelles de l'application mais également sur le code. Sans être exhaustif, les éléments suivants interviennent également dans la note finale : le bon usage du gestionnaire de versions, la qualité de la documentation, le respect des conventions du langage, le respect des consignes, la complexité, la lisibilité du code...

3 Conseils

Quelques conseils :

1. étant donné qu'il vous est demandé d'utiliser Git/Bazaar pour suivre votre travail, commencez par prendre connaissance avec l'outil avant de commencer l'implémentation ;
2. effectuez un commit de votre travail via Git/Bazaar aussi souvent que possible : dès qu'une fonctionnalité est implémentée et fonctionnelle (par exemple, passe avec succès vos tests). En cas d'erreur, cela vous permettra d'identifier et d'isoler plus rapidement l'origine du problème ;
3. prenez le temps de lire (et relire) le cours théorique afin de vous assurer de bien comprendre les notions théoriques intervenant dans ce projet ;
4. notez que le chapitre 4 du livre « The Theory of Relational Databases » de D. Maier contient des informations intéressantes sur les dépendances fonctionnelles, notamment sur une procédure efficace de calcul des conséquences logiques. Ce livre est disponible à l'adresse `http://web.cecs.pdx.edu/~maier/TheoryBook/MAIER/` ;
5. les modules Python suivants peuvent vous aider : `sqlite3` (accès à une base SQLite), `re` (expressions régulières), `argparse` (arguments en ligne de commande), `cmd` (interpréteur interactif), `pydoc` (génération de la documentation), `doctest` (tests unités auto-générés depuis la documentation) et `unittest` (tests unités).