UMONS
Université de Mons

Faculté des Sciences

Rapport de projet

SPJRUD vers SQL

S-INFO-102 Bases de données I (Partie C)

Réalisé par Pierre-Louis D'AGOSTINO
et Nicolas SOURNAC

Faculté
des Sciences

Année 2020-2021

ACADÉMIE
UNIVERSITAIRE
WALLONIE-
BRUXELLES

1 Introduction

Le but du projet est de créer en python, un outil permettant à un utilisateur d'entrer une expression SPJRUD afin qu'elle soit vérifiée puis compilée en SQL pour être exécutée sur une ou des tables d'une base de données.

2 Implémentation

Au niveau de l'implémentation en python, nous avons décidé de diviser le problème en 4 sous parties distinctes :

1. La création d'expressions SPJRUD
2. La création et la manipulation d'une base de donnée SQLite
3. La validation de la requête SPJRUD
4. La compilation et l'exécution de la requête SPJRUD en SQL

Pour le design de nos expressions SPJRUD, nous nous sommes inspirés du cours de génie logiciel, où nous avons étudié un exemple d'utilisation du design pattern composite afin de générer des expressions arithmétiques. C'est donc sur cette base que nous avons développé nos expressions SPJRUD. Ainsi, la super-classe Expr représente de manière abstraite une expression SPJRUD et ses classes enfants (Opération, Attribut, Relation, ListeAttribut, Cst) représentent les différents éléments qui peuvent composer une de nos expressions. Un atout majeur de notre outil est son mode console qui permet à l'utilisateur d'entrer en ligne de commande son expression tout en respectant une syntaxe définie. Afin d'implémenter cette fonctionnalité, nous avons créer un fichier UserInput.py où est regroupé l'ensemble des fonctions permettant la bonne analyse de l'expression entrée par l'utilisateur. Cette analyse repose sur une fonction récursive centrale : analyseInput() qui va parcourir la chaine de caractère et l'analyser récursivement en déterminant et en créant au fur et à mesure les opérations qui la composent.

Afin d'effectuer la validation d'une requête SPJRUD, chaque opération est munie d'une méthode vérif() et getCol(). La méthode vérif() se charge de vérifier que l'opération qui doit s'effectuer est correct en fonction du schéma de ses paramètres. La méthode getCol() retourne le schéma de l'opération lorsque celle-ci s'effectuera. Ces deux fonctions sont donc propres à chaque opérations SPJRUD. La super-classe Opération dispose d'une méthode recursive validation() qui va se charger d'effectuer les vérifications de toutes les sous-expressions que composent l'expression de base. Si une erreur est détectée, un message d'erreur personnalisé est alors affiché et une exception est lancée, ce qui stop le processus d'exécution du programme.

Une fois que la requête a été considérée comme valide par le processus de vérification, la traduction et l'exécution en sql de l'expression SPJRUD peut alors s'effectuer. Pour cela, chaque opération est munie d'une méthode op() qui va appeler elle même une fonction du fichier reqDefs() qui se chargera de créer une table temporaire composée du résultat de l'opération en fonction de ses paramètres dans la base de donnée et de retourner un objet de type Relation pointant vers cette table temporaire. La classe Opération dispose d'une méthode compute() qui va se charger de de traduire et exécuter récursivement chaque sous-expressions de l'expression de base.

En ce qui concerne la partie SQL, nous avons décidé de créer un fichier recup_data qui nous permet d'exécuter des méthodes sql rapidement, avec une syntaxe mieux adaptée aux objets que nous manipulions. Une solution pour réduire le nombre de lignes de code aurait été de créer une fonction plus générique pour exécuter n'importe qu'elle requête SQL en fonction de paramètres donnés.

3 Difficultés rencontrées

Durant ce projet, nous avons rencontré plusieurs difficultés ou bug. Par exemple dans la fonctionnalité d'analyse et de création d'une expression entrée par l'utilisateur dans la console, nous avons remarqué que lors de l'analyse du premier paramètre d'une opération de type JUD, il était impossible d'atteindre le second paramètre lorsque le premier paramètre était également une expression de type JUD. Après avoir effectué de nombreuses tentatives de correction, nous avons résolu ce bug en introduisant la fonction analyseLen().

Aussi, lors de la traduction et de la compilation, nous avons rencontré des difficultés dans la gestion de l'ordre des attributs des paramètres d'une différence ou d'une union. Afin de résoudre ce problème, nous avons contourné la requête (SELECT * FROM table1 EXCEPT SELECT * FROM table2) en lui donnant les attributs dans l'odre d'apparition (à l'aide de la méthode getColAndTypes() de la première table étant donné que nous savions au préalable que les noms et types d'attributs étaient corrects. Ce qui transforme la requête en (ex : SELECT Année, GP from table1 EXCEPT SELECT Année, GP from table2). Cette solution a également été appliquée pour l'opération d'union.

Enfin, la création de nombreuses tables temporaires lors de la traduction et compilation nous a forcé à un introduire une variable de classe "number" à la classe Opération afin de l'utiliser dans le nommage de ces tables temporaires.

4 Utilisation

L'utilisateur peut facilement entrer une expression SPJRUD car tout se fait via le mode console. Il suffit d'exécuter le fichier **(main.py)**, il vous sera ensuite demandé d'entrer une expression SPJRUD en respectant la syntaxe documentée dans chaque fonctions du fichier UserInput.py ou encore explicitée via des exemples dans le fichier texte Requêtes.txt.

Si la requête est considérée comme incorrect d'un point de vue sémantique, un message d'erreur personnalisé s'affichera et l'utilisateur sera invité un corriger sa requête en exécutant une nouvelle fois le fichier. Les erreur syntaxiques ne sont pas gérées. Si la requête est considérée comme correct, le résultat s'affichera alors.

Voilà un exemple de requête complexe :

Proj([ville],Select(année2,année,Join(Diff(Rel(villes) ;Rel(villes)) ;Rel(villes)))))