Rapport du projet de Base de Données I

DE COOMAN Thibaut STAQUET Gaëtan

25 décembre 2016

Table des matières

1 Introduction

Comme conseillé dans l'énoncé, nous avons employé le principe de l'AST pour implémenter ce vérificateur et traducteur.

Dans ce rapport, nous parlerons de l'utilisation de l'application, de la syntaxe à employer pour rentrer les commandes, de la façon dont l'AST est construit, de sa vérification et de sa traduction en SQL, ainsi que de la communication avec la base de données.

2 Utilisation

L'application permet de travailler sur une base de données déjà existante ou sur un schéma que l'utilisateur doit entrer. Une fois ceci fait, l'utilisateur peut entrer une requête SPJRUD. L'application vérifie cette requête et la traduit en SQL. Si l'utilisateur a demandé à employer une base de données, l'application exécute (via SQLite) la requête SQL et affiche le résultat. Dans tous les cas, l'application demande à l'utilisateur d'entrer une nouvelle requête (ou une ligne vide pour quitter l'application).

3 Syntaxe

3.1 SPJRUD

Les requêtes SPJRUD doivent respecter la syntaxe décrite dans la table suivante. E indique une relation algébrique et E' indique cette relation écrite dans la syntaxe demandée par l'application.

SPJRUD	**Application**
Relation R	Rel("R")
$\sigma_{A='a'}(E)$	Select(Eq("A", Cst('a')), E')
$\sigma_{A=B}(E)$	Select(Eq("A", Col("B")), E')
$\pi_X(E)$	Proj(["X_1", "X_2", "X_3", ..., "X_n"], E')
$E_1 \bowtie E_2$	Join(E'_1, E'_2)
$\rho_{A \to C}(E)$	Rename("A", "C", E')
$E_1 \cup E_2$	Union(E'_1, E'_2)
$E_1 - E_2$	Diff(E'_1, E'_2)

Table 1: SPJRUD vers la syntaxe de l'application

3.2 Schémas

La syntaxe à employer pour définir des schémas de relation est la suivante :

> "Nom de la relation", ("Nom de la colonne 1", "Type de la colonne (SQL types)", "Si la colonne peut contenir la valeur NULL ou non"), ("Nom de la colonne 2", ...), ...

Par exemple,

> "Notes", ("Nom", "VARCHAR(25)", False), ("Points", "INTEGER", False)

définirait la table suivante :

Nom	**Points**

Table 2: Table de données créée par un schéma

Nous n'avons pas permis de populer une table définie de cette façon.

4 Construction de l'AST

Nous supposons que nous travaillons avec la table de données suivante :

Name	**Country**	**Population**
Bruxelles	Belgique	184230
Paris	France	123456789

TABLE 3: Table Cities

Admettons que l'utilisateur veuille traduire la requête en SPJRUD suivante :

$$\rho_{Name \to City}(\pi_{Name}(\sigma_{Country='France'}(Cities) \ \cup \ \sigma_{Country='Belgique'}(Cities)))$$

Requête 1: Requête SPJRUD exemple

Cette requête devrait être encodée comme :

Rename("Name", "City", Proj(["Name"], Union(Select(Eq("Country", Cst("France")), Rel("Cities")), Select(Eq("Country", Cst("Belgique")), Rel("Cities")))))

Requête 2: Requête exemple

Pour pouvoir construire l'arbre correspondant à cette requête, nous avons décidé de procéder comme suit :

1. Vérifier les parenthèses et crochets
2. Décomposer la requête
3. Construire l'arbre nœud par nœud

La vérification des parenthèses et crochets est suffisament simple pour ne pas être expliquée ici.

4.1 Décomposition

Cette étape crée une liste de listes et/ou de chaînes de caractères. Par exemple, *Select(Eq("A", Cst(a)), Rel("R"))* donne la décomposition *["Select", ["Eq", ["A", "Cst", ["a"]], "Rel", ["R"]]]*
La Requête exemple donnerait ainsi la décomposition :

["Rename", ["Name", "City", "Proj", [["Name"], "Union", ["Select", ["Eq", ["Country", "Cst", ["France"]], "Rel", ["Cities"]], "Select", ["Eq", ["Country", "Cst", ["Belgique"]], "Rel", ["Cities"]]]]]]

Décomposition 1: Décomposition de la requête exemple

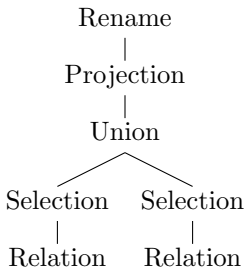
4.2 Construction

Cette décomposition permet de construire un algorithme récursif pour construire l'arbre. Appelons-le *build_AST*. Cet algorithme prend en paramètre la sous-liste qui doit être traitée (le cas de base étant la Relation). Ainsi, la décomposition *["Select", ["Eq", ["A", "Cst", ["a"]], "Relation", ["R"]]]* donnerait l'exécution suivante :

1. `build_AST`(["Select", ["Eq", ["A", "Cst", ["a"]], "Relation", ["R"]]])
   (a) `build_AST`(["Relation", ["R"]])

Lorque *build_AST* rencontre un nom d'opération, il récupère les valeurs nécessaires à cette opération. Par exemple, l'Union demande deux sous-requêtes tandis que la Relation demande le nom de la relation.

De la liste 1, nous pouvons construire l'arbre suivant :



AST 1: AST de la décomposition de la requête exemple

Le nom indiqué dans les nœuds est le nom de la classe employée.

5 Vérification de l'AST

Une fois l'arbre construit, nous pouvons commencer à vérifier s'il est correct. Par *correct*, nous entendons qu'il répond aux exigences et à la logique de l'algèbre relationnel (que les schémas soient respectés, ...). Pour ce faire, chaque classe définit une fonction *check*. Si l'opération n'est pas une Relation, cette méthode appelle le *check* du/des nœud(s) enfant(s). Cette méthode construit également le schéma qui découle de celui du/des enfant(s) et de l'opération (par exemple, la sélection ne modifie pas le *sorte* mais la projection peut retirer des colonnes).

Nous nous sommes basés sur les définitions du cours théorique pour implémenter cette méthode.

Si une erreur est détectée, une Exception décrivant le problème est lancée et récupérée par la fonction *main*. Ainsi, l'AST est correct et validé si aucune exception n'a été lancée.

6 Traduction de l'AST

Une fois notre AST validé, nous pouvons le traduire en une requête SQL. Pour ce faire, nous avons une classe *SQLRequest* qui contient les informations sur la clause FROM, les conditions, les colonnes, les alias, ... Les instances de cette classe sont construites par la méthode *translate* de *Operation*. Comme pour *check*, cette méthode demande d'abord aux opérations enfants de créer leur traduction.

Dans le tableau suivant, E indique une requête en SPJRUD tandis que E' indique la traduction de cette requête.

Opération (SPJRUD)	SQL	SQLRequest
Relation R	SELECT * FROM R	La clause FROM vaut R et la liste des colonnes vaut celle de la relation
$\sigma_{A="a"}E$	E' WHERE A="a"	On ajoute la condition $A="a"$
$\sigma_{A=B}E$	E' WHERE A=B	On ajoute la condition $A=B$
$\pi_X E$	SELECT X FROM ...	On ne garde que les colonnes X
$E_1 \bowtie E_2$	SELECT * FROM (E'_1 INNER JOIN (E'_2) USING (col_1, col_2, ...)	On met les colonnes communes aux deux schémas dans USING
$\rho_{A\to B}E$	SELECT A AS B, autres colonnes FROM ...	On ajoute un alias sur A et on remplace toutes les occurences de A par B dans les conditions
$E_1 \cup E_2$	E'_1 UNION E'_2	On met la clause UNION entre les deux sous-requêtes
$E_1 - E_2$	E'_1 EXCEPT E'_2	On met la clause EXCEPT entre les deux sous-requêtes

TABLE 4: SPJRUD vers SQL (avec les modifications de l'application)

Pour des raisons de simplicité de code, nous avons décidé d'indiquer à chaque fois explicitement les colonnes, même quand il n'y a pas de projection. Le programme traduira *Rel("A")* (en supposant que $sorte(A) = B, X, Y$ en *SELECT B, X, Y FROM A*.

Ainsi, notre requête 1 donnerait la requête en SQL :

```
SELECT Name AS City FROM ((SELECT Name, Country,
Population FROM Cities WHERE Country="France") UNION
(SELECT Name, Country, Population FROM Cities WHERE
Country="Belgique"))
```

Les deux membres de l'Union auraient pu être réunis en une seule sous-requête en joignant les deux conditions par un *OR* mais, dans l'optique de garder le code simple et efficace dans toutes les conditions, nous avons préféré laisser l'Union telle quelle.

7 Problèmes rencontrés

7.1 Construction de l'AST

Notre premier problème a été de constuire efficacement l'AST (ce qui est, bien sûr, fortement lié à l'énoncé) et de répondre aux questions suivantes :

— Quelle est la syntaxe qu'on peut traiter ?
— Comment vérifier la syntaxe rentrée ?
— Comment vérifier que l'arbre est correct ?
— Comment traduire ?

Les réponses à ces questions ont été expliquées dans les sections précédentes.

7.2 Organisation

Etant donné que nous avons eu un autre projet, nous avons dû nous répartir le travail afin d'être capable de finir le projet.

7.3 Emploi de SQLite

Nous ne connaissions pas SQLite avant ce projet. Il nous a donc fallu apprendre à correctement communiquer avec ctte bibliothèque, en faisant des essais grâce à l'interpréteur Python.

De plus, ce projet est le premier que nous avons fait en Python, mais cela ne nous a pas posé de problème.