

Rapport du projet de Base de Données I

DE COOMAN Thibaut STAQUET Gaëtan

21 décembre 2016

Table des matières

1	Introduction	1
2	Utilisation	2
3	Syntaxe	2
3.1	SPJRUD	2
3.2	Schémas	2
4	Construction de l'AST	3
4.1	Décomposition	3
4.2	Construction	3
5	Vérification de l'AST	4
6	Traduction de l'AST	4

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.

Nous supposons que nous travaillons dans la base de données suivante :

Name	Country	Population
Bruxelles	Belgique	184230
Paris	France	123456789

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)
$\rho_{A \rightarrow C}(E)$	Rename("A", "C", E')
$E_1 \cup E_2$	Union(E'_1 , E'_2)
$E_1 - E_2$	Diff(E'_1 , E'_2)

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
-----	--------

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

4 Construction de l'AST

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

Requête SPJRUD 1. $\rho_{Name \rightarrow City}(\pi_{Name}(\sigma_{Country='France'}(Cities) \cup \sigma_{Country='Belgique'}(Cities)))$

Cette requête devrait être encodée comme :

Requête 1. *Rename("Name", "City", Proj(["Name"], Union(Select(Eq("Country", Cst("France")), Rel("Cities")), Select(Eq("Country", Cst("Belgique")), Rel("Cities")))))*

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 suffisamment 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 1 donnerait ainsi la décomposition :

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

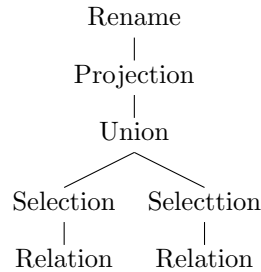
4.2 Construction

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"]])*

Lorsque *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 4.1, nous pouvons construire l'arbre suivant :



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).

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
$\sigma_{A="a"}E$	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
$E_1 \bowtie E_2$??	??
$\rho_{A \rightarrow B} E$	SELECT A AS B, autres colonnes FROM ...	On ajoute un alias sur A et on remplace toutes les occurrences de A par B dans les conditions
$E_1 \cup E_2$??	??
$E_1 - E_2$??	??