Projet de « Bases de Données I »
Compilation SPJRUD vers SQL

Pierre Hauweele

2015 – 2016

1 Description du projet

1.1 Introduction

L'objectif de ce projet est d'implémenter, en Python, un outil de compilation (traduction) de requêtes SPJRUD vers des requêtes SQL.

De base, cet outil doit fournir un moyen de compiler une expression écrite en SPJRUD vers l'expression d'une requête SQL et de vérifier que l'expression SPJRUD est correcte (arité, noms des attributs existants, ...). Cet outil doit également pouvoir être utilisé pour facilement exécuter ces requêtes sur une base de données SQLite [1]. Des fonctionnalités supplémentaires peuvent être implémentées.

Vous documenterez également vos choix d'implémentation dans le code et, si vous le souhaitez, dans un document séparé.

1.2 Représentation des expressions SPJRUD

Une expression SPJRUD sera représentée sous forme d'un AST (Abstract Syntax Tree [2]). Un AST représente une expression syntaxique sous forme d'un arbre ayant pour nœuds les opérateurs de l'expression et chaque nœud ayant comme fils ses arguments. Par exemple, l'expression

$$\pi_{\mathsf{Population}}((\rho_{\mathsf{Name}\mapsto\mathsf{Capital}}(\mathsf{Cities})) \bowtie (\sigma_{\mathsf{Country}=\text{“Mali”}}(\mathsf{CC}))), \tag{1}$$

peut être représentée par l'AST de la figure 1, où $\mathsf{Cities}_{\mathsf{rel}}$, est la relation Cities et $\mathsf{Country}_{\mathsf{attr}}$ est l'attribut $\mathsf{Country}$.

Chaque nœud de cet arbre peut être représenté en Python par un objet ayant comme variables de classe ses paramètres. Un nœud de l'AST peut être créé par le constructeur de l'objet correspondant. Par exemple, l'AST correspondant à l'expression SPJRUD (1) pourrait être construit par l'expression Python suivante :

```
Proj(['Population'],
     Join(Rename('Name', 'Capital', Rel('Cities')),
          Select(Eq('Country', Cst('Mali')),
                 Rel('CC'))))
```

1. Python SQLite3 : https://docs.python.org/2/library/sqlite3.html
2. AST wikipedia : https://en.wikipedia.org/wiki/Abstract_syntax_tree

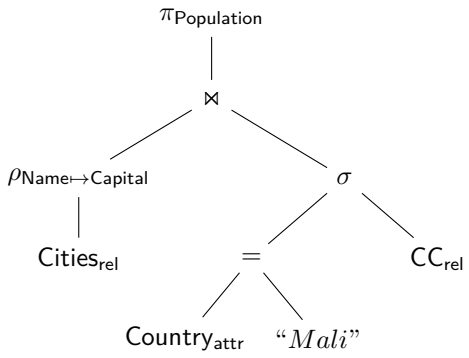

FIGURE 1 – AST pour l'expression (1)

Ici, certains choix sont pris, comme le fait que les attributs sont directement représentés par des chaînes de caractères qui ne sont pas encapsulées dans, par exemple, un objet `Attr`. Votre implémentation ne doit pas nécessairement satisfaire à la syntaxe présentée ici. Par exemple, l'expression

```
Select(Eq('Country', Cst('Mali')), Rel('CC'))
```

aurait pu être, sous certaines réserves, écrite de la manière suivante :

```
Select('Country', '=', Cst('Mali'), Rel('CC'))
```

On considère qu'un opérateur est une expression SPJRUD et que celui-ci s'applique sur une autre expression SPJRUD. Le traitement d'une expression SPJRUD s'effectue donc de manière récursive. Le cas de base de cette récursion est un opérateur qui ne compose pas avec une sous-requête. Par exemple, l'opérateur `Rel`, fait directement référence à une relation (c'est ce qu'on appelle un atome de la syntaxe). Tous les objets représentant les opérateurs peuvent donc hériter d'un objet `Expr`. L'opérateur de différence $A - B$ pourrait par exemple être représenté par un objet `Diff` stockant les objets `Expr` de gauche et de droite de l'expression `Diff(A, B)`.

Vous êtes libre d'utiliser une représentation différente de celle donnée ici. Cependant, veillez à justifier ces choix dans votre documentation.

1.3 Relations

Les requêtes SPJRUD s'exécutent sur des relations. Une relation suit un certain schéma et est constituée de tuples. Dans la librairie développée dans ce projet, les requêtes SPJRUD seront exécutées sur une base de données SQLite. Les relations référencées dans une requête correspondent aux tables de la base de données SQLite sur laquelle elle sera exécutée. Par exemple, si on demande à compiler la requête (1) sur la base de données SQLite `db`, on s'attend (et la librairie devra le vérifier) à ce que la base de données `db` comprenne les tables `Cities` ainsi que `CC`. Leur schéma dans la base de données SQLite détermine leur schéma attendu pour la requête SPJRUD. De plus, le schéma d'une table en SQLite donne des informations sur le type de ses données. La librairie devrait donc vérifier non seulement la compatibilité de l'arité des relations, des noms

d'attributs, mais également des types de données des attributs. Par exemple, soit `Cities` la table créée par la requête SQL suivante :

```
CREATE TABLE ‘Cities‘ (
       ‘Name‘       TEXT,
       ‘Country‘    TEXT,
       ‘Population‘ NUMERIC,
       PRIMARY KEY(Name,Country)
);
```

La librairie devrait considérer la requête $\sigma_{\mathsf{Name}=\mathsf{Population}}(\mathsf{Cities})$ comme invalide, puisque les données des colonnes Name et Population ne sont pas du même type.

1.4 Validation des requêtes

Comme mentionné précédemment, votre librairie doit pouvoir vérifier qu'une requête SPJRUD est valide avant de la traduire vers une requête SQL.

On considère que la validation s'effectue toujours en connaissant le schéma de la base de données sur laquelle on exécute la requête. La description du schéma de la base de données peut être déduite de la base données SQLite, ou bien être elle même décrite par une structure en Python (de manière à ce qu'une expression puisse être compilée sur un *schéma* de base de données sans nécessairement la lier à une base de données SQLite physique).

Il serait intéressant que, dans le cas où le requête est invalide, la librairie donne un renseignement sur l'origine du problème et une explication. Par exemple, la validation de la requête suivante :

```
Select(Eq('Country', Cst('Mali')),
       Diff(Rel('Cities'),
            Proj(['Name', 'Country'], Rel('Cities'))))
```

pourrait donner un message du type suivant :

```
Invalid expression.
The (sub-)expression
  Diff(Rel('Cities'), Proj(['Name', 'Country'], Rel('Cities')))
is invalid because the schema of
  Rel('Cities')
which is
  'Name' TEXT,
  'Country' TEXT,
  'Population' NUMERIC
is not the same as the one from
  Proj(['Name', 'Country'], Rel('Cities'))
which is
  'Name' TEXT,
  'Country' TEXT
```

Notez bien la séparation entre la *validation* de la requête, et sa *compilation* à proprement dit. N'oubliez pas de vérifier les types de données dans les schémas.

1.5 Traduction et exécution des requêtes

Une fois validée, une expression SPJRUD pourra être traduite vers une expression SQL. On peut considérer qu'une expression SQL est une chaîne de caractères représentant la requête telle qu'elle sera passée en paramètres à la méthode `execute` de SQLite. Cette requête doit ensuite pouvoir être exécutée sur une base de données SQLite réelle.

1.6 Fonctions utilitaires

Pensez à fournir des fonctions utilitaires permettant notamment de :
- convertir une expression SPJRUD en une chaîne de caractères (vous en aurez besoin pour afficher les messages d'erreur) ;
- créer une nouvelle table à partir du résultat d'une requête ;
- afficher le résultat d'une requête à l'écran.

2 Organisation

Le projet est réaliser individuellement ou par groupe de deux, au choix. Il vous est imposé d'utiliser Bazaar pour organiser votre travail. Veillez à documenter votre travail et à fournir un fichier `README` ou un document au format PDF expliquant vos choix d'implémentation, les fonctionnalités supplémentaires, ainsi que les difficultés rencontrées et les solutions apportées ou envisagées. Vous pouvez également y ajouter une documentation de l'utilisation de votre librairie et d'éventuelles remarques sur votre travail.

Le travail doit être livré sous la forme d'une archive (`.zip` ou un format libre) portant, en majuscules et sans accents, les noms et prenoms des membres du groupe séparés par un tiret. L'archive doit contenir votre application, le fichier `README` ou le document PDF correspondant et le répertoire `.bzr`. Si vous souhaitez fournir d'autres éléments (documentation auto générée, fichiers tests, etc.), placez les dans un répertoire nommé `misc`. N'oubliez pas de synchroniser votre travail via Bazaar et de valider vos derniers changements avant de remettre l'archive.

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...

Veuillez communiquer la composition de votre groupe (individuel ou en binôme) à l'adresse e-mail pierre.hauweele@umons.ac.be pour le **vendredi 20 novembre**. L'archive devra être remise via la plateforme Moodle pour le **lundi 14 décembre à minuit au plus tard**.