Introduction

Membres du groupe : *COLLIN Florent, CASSART Justin*

Ce rapport regroupe le guide d'utilisation de la librairie et les choix de modélisation. Le guide d'utilisation explique
en détail comment utiliser la librairie. Quant aux choix de modélisation, ils reprennent l'explication des choix effectués lors du développement.

Guide-Utilisateur

Cette section a pour but d'expliquer et de montrer le fonctionnement de la librairie nommée *algebraToSql*.

Prérequis

Pour pouvoir utilisé la librairie *algebraToSql*, la librairie *pandas* doit être installé. Cette librairie a été utilisée dans le projet afin d'améliorer l'affichage de la réponse des requêtes sur une base de données SQLite3. Un fichier *requirements.txt* est disponible afin de faciliter l'installation de *pandas*.

Installation sur Windows:

```
pip install -r requirements.txt
```

Installation sur les distributions Linux:

```
pip3 install -r requirements.txt
```

Création de requêtes SPJRUD

Afin de créer des requêtes en algèbre SPJRUD, il est nécessaire d'importer la librairie *algebraToSql*.

```
from algebraToSql import *
```

Chaque requête nécessite de travailler sur une table. La class *Table* permet de créer une table, elle

prend en paramètre le nom de cette table (String) et son schéma (dict).
Le schéma d'une table doit être un dictionnaire ayant comme clés des chaînes de caractères correspondant aux noms des attributs et comme valeurs, le type de l'attribut. Le type d'un attribut doit être parmi les types suivants :

- *TEXT*
- *INTEGER*
- *REAL*

```
table = Table("employee", {"name":"TEXT", "number":"INTEGER", "salary":"REAL"})
```

Il est aussi possible de créer un schéma de base de données qui contient des tables.

```
employee = Table("employee", {"name":"TEXT", "number":"INTEGER", "salary":"REAL"})
departement = Table("departement", {"name":"TEXT", "loc":"TEXT"})
db = DBSchema()
db.addTable(employee)
db.addTable(departement)
#La méthode get(name) permet de récupérer une table grâce à son nom.
employee = db.get("employee")
```

La classe Rel

La classe *Rel* est la classe mère, elle représente une relation.

```
table = Table("employee", {"name":"TEXT", "number":"INTEGER", "salary":"REAL"})
rel = Rel(table) #représente la relation associée à la table "employee"
```

La classe Select

La classe *Select* représente la sélection dans l'algèbre SPJRUD. Le constructeur de cette classe prend en paramètre une opération (=,>,>=, <, <=) et une relation.

```
table = Table("employee", {"name":"TEXT", "number":"INTEGER", "salary":"REAL"})
s = Select(Eq("name", Const("Jean")), Rel(table))#Const représente une constante
s = Select(Eq("number", Attribute("salary")), Rel(table))
#Attribute représente un attribut d'une table
s = Select(Greather("salary", Const(1500)), Rel(table))
s = Select(GreatherOrEqual("salary", Const(1000)), Rel(table))
s = Select(Less("salary", Const(2000)), Rel(table))
s = Select(LessOrEqual("salary", Const(1500)), Rel(table))
```

La classe Proj

La classe *Proj* représente la projection dans l'algèbre SPJRUD. Le constructeur de cette classe prend en paramètre une liste des attributs sur lesquels projeter et une relation.

```
table = Table("employee", {"name":"TEXT", "number":"INTEGER", "salary":"REAL"})
p = Proj(["name", "salary"], Rel(table))
```

La classe Join

La classe *Join* représente la jointure dans l'algèbre SPJRUD. Le constructeur de cette classe prend en paramètre deux relations.

```
employee = Table("employee",
          {"name":"TEXT", "number":"INTEGER", "salary":"REAL", "deptName":"TEXT"})
departement = Table("departement", {"deptName":"TEXT", "loc":"TEXT"})
j = Join(Rel(employee), Rel(departement))
```

La classe Rename

La classe *Rename* représente le renommage dans l'algèbre SPJRUD. Le constructeur de cette classe prend en paramètre l'attribut à renommer (String), le nouveau nom de l'attribut (String) et une relation.

```
table = Table("employee", {"name":"TEXT", "number":"INTEGER", "salary":"REAL"})
r = Rename("name", "Nom", Rel(table))
```

La classe Union

La classe *Union* représente l'union dans l'algèbre SPJRUD. Le constructeur de cette classe prend en paramètre deux relations ayant les mêmes attributs

```
employee = Table("employee", {"name":"TEXT", "number":"INTEGER", "salary":"REAL"})
employee2 = Table("employee", {"name":"TEXT", "number":"INTEGER", "salary":"REAL"})
u = Union(Rel(employee), Rel(employee2))
```

La classe Diff

La classe *Diff* représente la différence dans l'algèbre SPJRUD. Le constructeur de cette classe prend en paramètre deux relations ayant les mêmes attributs.

```
```

```
employee = Table("employee", {"name":"TEXT", "number":"INTEGER", "salary":"REAL"})
employee2 = Table("employee", {"name":"TEXT", "number":"INTEGER", "salary":"REAL"})
d = Diff(Rel(employee), Rel(employee2))
```

Combinaison de requêtes SPJRUD

L'atout majeur de l'algèbre SPJRUD est la combinaison de requêtes. Il est donc possible de le faire facilement.

```
#Exemple de combinaison de requêtes SPJRUD
employee = Table("employee",
    {"name":"TEXT", "number":"INTEGER", "salary":"REAL", "deptName":"TEXT"})
departement = Table("departement", {"name":"TEXT", "loc":"TEXT"})
S = Select(Eq("name", Const("Jean")), Rel(employee))
P = Proj(["name", "salary"], S)

R = Rename("name", "deptName", Rel(departement))
J = Join(R, Rel(employee))
```

Conversion en requêtes SQL

Chaque requête en algèbre SPJRUD peut être convertie en requête SQL grâce à la fonction toSql().

```
table = Table("employee", {"name":"TEXT", "number":"INTEGER", "salary":"REAL"})
p = Proj(["name", "salary"], Rel(table))
p_sql = p.toSql()
# p_sql = "select distinct name, salary from employee"
```

Choix d'implémentation

Classe DBSchema

La classe DBSchema permet de stocker un schéma de base de données. Sa méthode addTable(table) peut être utilisée pour ajouter au schéma de base de données une table. Et la méthode get(name) permet de récupérer la table correspondant au nom donné en paramètre.

Classe Table

Cette classe permet de gérer un seul schéma de table.

Pour implémenter cette classe, deux solutions étaient possibles.

- Utiliser une liste pour les attributs et un autre pour les types des attributs

- Utiliser un dictionnaire

L'utilisation d'un dictionnaire évite pas mal de problèmes comparés aux listes, notamment lorsque qu'on cherche le type d'un attribut on peut directement le chercher dans le dictionnaire sans devoir chercher l'index de l'attribut.

Méthode checkType

Cette méthode permet de vérifier si le type d'une valeur est bien égal au type d'un attribut.
Cette méthode est implémentée, car *Python* et *SPJRUD* ne prennent pas forcément le même nom pour un certain type. Par exemple une chaîne de caractère en *Python* est du type *str* or en *SPJRUD* le type sera *TEXT*. Par conséquent il a été décidé d'implémenter cette méthode afin de faciliter les vérifications de types.

Classe Rel

Il a été décidé d'implémenter une super classe *Rel* afin de réduire le coût d'implémentation des autres classes. De ce fait tout ce qui est commun aux sous-classes, se trouvant être toutes les requêtes SPJRUD, c'est-à-dire *Select, Proj, Join, Rename, Union, Diff*, est implémenté dans cette superclasse comme la vérification de l'existence des tables.

Classe Operation

Cette classe, qui est utilisée pour la sélection, permet de facilement écrire des opérations. De telle manière la classe Select ne prend que deux paramètres, une opération et une relation (cf. guide-utilisateur). Les classes *Eq, Greather, GreatherOfEqual, Less, LessOrEqual* héritent de la classe Operation et permettent d'écrire les différentes comparaisons possibles.

Classe Const/Attribute

Cette classe permet simplement d'utiliser un(e) constante/attribut dans une méthode SPJRUD. De cette façon il est plus simple de gérer si le paramètre est bien un(e) constante/attribut sans devoir se soucier de tous les types possibles.

Classe SQLite

C'est cette classe qui exécute la requête SQL. Il suffit de lui passer en paramètre le nom du fichier contenant la base de données. Connaissant ce fichier, la classe est capable de déterminer le DBSchema. De ce fait pour obtenir une table l'utilisateur n'a tout simplement qu'à passer en argument le nom de la table qu'il souhaite utiliser depuis le DBSchema de cette classe.