Introduction

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.

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 utilisé 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 facilité l'installation de *pandas*.

Installation sur Windows:

```
python pip -m 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 associé à&amp;nbsp;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 constanste
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(LessOrEqal("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"})
j = Join(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"})
j = 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 est égal à "select distinct name, salary from employee"
```

Utilisation des requêtes SPJRUD sur une base de données SQLite3

Afin d'utiliser les requêtes SPJRUD sur une base de données SQLite3 il est nécessaire de créer une instance de la classe SQLite. Cette classe permet d'effectuer des requêtes sur une base de données déjà existante et de récupérer un DBSchema qui représente les différentes tables de cette base de données.

```
s = SQLite("test.db") # "test.db" est le nom du fichier contenant la base de données
#On récupère un object DBSchema qui représente le schéma de la base de données
db = s.dbSchema
# On récupère la table "employee" dans le schéma de base de données
# et on crée une projection sur "name"
request = Proj(["name"], Rel(db.get("employee")))
# On peut ensuite exécuter cette requête
result = s.execute(request)
# On peut aussi juste afficher le résultat de la requête
s.execute(request, _print=True)
```