Вступительное задание

Реализация некоторого подобия языка управления данными в коллекции. Основные команды, которые должны поддерживаться это вставка элементов в коллекцию, удаление элемента из коллекции, поиск элементов в коллекции, изменение элементов в коллекции.

Структура коллекции заранее определена.

Описание задачи:

Коллекция данных в данной задаче это структура представляющая собой некоторую таблицу данных, в которой есть наименования колонок и каждая строчка в таблице это элемент коллекции.

Необходимо реализовать метод, который на вход получает команду в виде строки (требования к формату будет ниже). Команда должна выполнять четыре основные операции вставка, изменение, поиск и удаление элементов из коллекции данных.

Также при изменении, удалении и поиске должны поддерживаться условия выборки из коллекции (ниже они будут представлены).

На выход список элементов в коллекции, которые были найдены, либо которые были изменены, либо которые были добавлены, либо которые были удалены.

Требования к задаче:

1. Требования к коллекции:

Коллекция представляет из себя таблицу в виде List<Map<String,Object>> . Где List это список строк в таблице. Map это значения в колонках, где ключом Map является наименование колонки в виде строки, а значением Map является значение в ячейке таблицы (допустимые типы значений в ячейках: Long, Double, Boolean, String).

Например таблица с пользователями вида:

Таблица 1: Список пользователей.

id	lastName	age	cost	active
1	Петров	30	5.4	true
2	Иванов	25	4.3	false

Может быть представлена в виде:

```
Map<String,Object> row1 = new HashMap<>();
row1.put("id",1);
row1.put("lastName","Петров");
row1.put("age",30);
row1.put("cost",5.4);
row1.put("active", true);

Map<String,Object> row2 = new HashMap<>();
row2.put("id",2);
row2.put("lastName","Иванов");
row2.put("age",25);
row2.put("cost",4.3);
row2.put("active", false);

List<Map<String,Object>> data = new ArrayList<>();
data.add(row1);
data.add(row2);
```

Уникальность значений в ячейках не проверяется, т.е. может быть две записи с id=1 или две записи с lastName=”Иванов”. Также некоторые значения могут быть пустыми, но все значения в ячейках в одной строчке пустыми быть не могут.

2. Требования к формату запроса с командой, поступающей на вход:

Возможные команды:

a. **INSERT** - вставка элемента в коллекцию,
b. **UPDATE** изменение элемента в коллекции,
c. **DELETE** - удаление элемента из коллекции,
d. **SELECT** - поиск элементов в коллекции.

Возможные операторы сравнения:

Перед началом оператора сравнения должен стоять оператор WHERE - оператор, который говорит, что команда должна выполняться с условием выборки.

Таблица 2: Список операторов сравнения.

№	Оператор	Назначение	Примеры
1	=	Оператор равно. Возвращает true если два значения и их типы	5=5 вернет true,

		равны между собой, иначе false. Применяется на типы данных: Boolean, String, Long, Double	‘test’=’test’ вернет true 5=’test’ вернет false ‘5’=5 вернет false true=true вернет true false=false вернет true 5=false вернет false 5=5.0 вернет false
2	!=	Оператор не равно. Возвращает true если два значения или их типы не равны между собой, иначе false. Применяется на типы данных: Boolean, String, Long, Double	5!=5 вернет false, ‘test’!=’test’ вернет false 5!=’test’ вернет true ‘5’!=5 вернет true true!=true вернет false false!=false вернет false 5!=false вернет true 5!=5.0 вернет true
3	like	Оператор поиска удовлетворяющих шаблону. Шаблон может быть следующих видов: ‘строка’, ‘строка%’, ‘%строка%’, ‘%строка’. Применяется на типы данных: String	‘test’ like ‘test’ вернет true ‘test’ like ‘tEst’ вернет false ‘test’ like ‘test%’ вернет true ‘testAD’ like ‘test%’ вернет true ‘ADtestAD’ like ‘test%’ вернет false ‘test’ like ‘%test’ вернет true ‘ADtest’ like ‘%test’ вернет true ‘ADtestAD’ like ‘%test’ вернет false ‘ADtestA’ like ‘%test%’ вернет true

			‘test’ like ‘%test%’ вернет true ‘ADteAst’ like ‘%test%’ вернет false
4	ilike	Оператор поиска удовлетворяющих шаблону. Выполняется аналогично оператору like, но без учета регистра. Шаблон может быть следующих видов: ‘строка’, ‘строка%’, ‘%строка%’, ‘%строка’. Применяется на типы данных: String	‘test’ like ‘test’ вернет true ‘test’ like ‘tEst’ вернет true ‘test’ like ‘tEst%’ вернет true ‘testAD’ like ‘test%’ вернет true ‘ADtestAD’ like ‘test%’ вернет false ‘test’ like ‘%test’ вернет true ‘ADtest’ like ‘%test’ вернет true ‘ADtestAD’ like ‘%test’ вернет false ‘ADtestA’ like ‘%test%’ вернет true ‘test’ like ‘%test%’ вернет true ‘ADteAst’ like ‘%test%’ вернет false
5	**>=**	Оператор больше равно. Применяется для типов: Long, Double	0>=0.1 вернет false 0>=0 вернет true 0>=-1 вернет true 0>=’0’ вернет ошибку 1>=true вернет ошибку 0.1>=1 вернет false
6	**<=**	Оператор меньше равно. Применяется для типов: Long, Double	0<=0.1 вернет true 0<=0 вернет true 0<=-1 вернет

			false 0<='0' вернет ошибку 1<=true вернет ошибку 0.1<=1 вернет true
7	**<**	Оператор меньше. Применяется для типов: Long, Double	0<0.1 вернет true 0<0 вернет false 0<-1 вернет false 0<'0' вернет ошибку 1<true вернет ошибку 0.1<1 вернет true
8	**>**	Оператор больше. Применяется для типов: Long, Double	0>0.1 вернет false 0>0 вернет false 0>-1 вернет true 0>'0' вернет ошибку 1>true вернет ошибку 0.1>1 вернет false

Логические операторы:

Таблица 3: Список логических операторов.

№	Оператор	Назначение	Примеры
1	AND	Логический оператор “И”.	0>=0 AND ‘test’ like ‘test’ вернет true 0>0 AND ‘test’ like ‘test’ вернет false
2	OR	Логический оператор “ИЛИ”.	0>0 OR ‘test’ like ‘test’ вернет true 0>0 OR ‘tEst’ like ‘test’ вернет false 0>=0 OR ‘test’ like ‘test’ вернет true

* Поддержку скобок реализовывать не нужно.

Вставка и изменение ячеек:

Для вставки новых значений или изменение существующих необходимо передавать в строку запроса оператор VALUES и далее имя колонки = значение (через запятую).

Примеры запросов:

1. **Пример 1**: Пример добавления строки в коллекцию:

На вход подается запрос:

INSERT VALUES ‘lastName’ = ‘Федоров’ , ‘id’=3, ‘age’=40, ‘active’=true

Что означает, что должна быть добавлена запись в коллекцию вида (на примере Таблица 1):

Таблица 1.1: Результат добавления

id	lastName	age	cost	active
1	Петров	30	5.4	true
2	Иванов	25	4.3	false
3	Федоров	40		true

```
Map<String,Object> row3 = new HashMap<>();
row3.put("id",3);
row3.put("lastName","Федоров");
row3.put("age",40);
row3.put("active", true);

data.add(row3);
```

На выход из метода: List<Map<String,Object>> в которой содержится row3 (со всеми колонками, в которых есть значение).

2. **Пример 2**: Пример изменения строки в коллекции:

На вход подается запрос:

UPDATE VALUES ‘active’=false, ‘cost’=10.1 where ‘id’=3

Что означает, что должны быть изменены строки, у которых колонка id равна 3 (на примере Таблица 1.1):

Таблица 1.2: Результат изменения

id	lastName	age	cost	active
1	Петров	30	5.4	true
2	Иванов	25	4.3	false
3	Федоров	40	10.1	false

```
row3.put("cost",10.1);
row3.put("active", false);
```

На выход из метода: List<Map<String,Object>> в которой содержится row3 с измененными данными (со всеми колонками, в которых есть значение).

3. **Пример 3**: Пример изменения строк в коллекции:

На вход подается запрос:

UPDATE VALUES ‘active’=true where ‘active’=false

Что означает, что должны быть изменены все строки, у которых колонка active равна false (на примере Таблица 1.2):

Таблица 1.3: Результат изменения

id	lastName	age	cost	active
1	Петров	30	5.4	true
2	Иванов	25	4.3	true
3	Федоров	40	10.1	true

```
row3.put("active", true);

row2.put("active", true);
```

```
На выход из метода: List<Map<String,Object>> в которой содержится row3 с измененными данными (со всеми колонками, в которых есть значение).
```

4. **Пример 4**: Пример поиска строк в коллекции:

На вход подается запрос:

SELECT WHERE ‘age’>=30 and ‘lastName’ ilike ‘%п%’

Что означает, что необходимо найти все строки, у которых значение в колонке age больше или равно 30 и значение в колонке lastName содержит букву ‘п’ без учета регистра (на примере Таблица 1.3):

```
На выход из метода: List<Map<String,Object>> в которой содержится row1, поскольку только первая строчка попадает под условия выборки.
```

5. **Пример 5**: Пример удаления строк в коллекции:

На вход подается запрос:

DELETE WHERE ‘id’=3

Что означает, что необходимо удалить все строчки, в которых в колонке id значение равно 3. (на примере Таблица 1.3):

Таблица 1.4: Результат удаления

id	lastName	age	cost	active
1	Петров	30	5.4	true
2	Иванов	25	4.3	true

```
На выход из метода: List<Map<String,Object>> в которой содержится row3, поскольку только третья строчка попадает под условия выборки.
```

Команды DELETE, SELECT, UPDATE могут выполняться без условия WHERE. В этом случае все записи должны быть получены, изменены или удалены.

Команда INSERT всегда выполняется без оператора WHERE.

Требования к реализации:

1. В этом пакете должен лежать класс `JavaSchoolStarter`, в котором есть метод execute на вход передается строка, а на выход `List<Map<String,Object>>`. Дефолтный конструктор без аргументов обязательно должен быть (он указан в примере).

   Пример:

```
Package test;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

public class Test {
  //Дефолтный конструктор
public Test (){

}

  //На вход запрос, на выход результат выполнения запроса
   public List<Map<String,Object>> execute(String request) throws Exception {
     //Здесь начало исполнения вашего кода

     //  return new ArrayList<>();
   }
}
```

4. Скобки в условиях реализовывать не нужно, но если сделаете - будет плюсом.
5. Проверяться будет в первую очередь корректность исполнения запроса, потом уже только сама реализация и код.
6. Если не все реализовано - сообщить об этом.
7. Валидацию запроса проверять не нужно, но если сделаете - будет плюсом
8. Если в запросе есть наименование колонки, которой нет в таблице - выдать Exception. Также если в сравнении участвует тип, который не поддерживается данным оператором выкинуть также Exception (например 'lastName'>10
9. Наименование колонок и структуру данных берем из примера (см. Таблица 1). Колонка id имеет тип Long, колонка lastName имеет тип String, колонка cost имеет тип Double, колонка age имеет тип Long, колонка active имеет тип Boolean.
10. Коллекция на старте должна быть пустой (изначально не заполнять).
11. Лишние пробелы игнорируются, но в значении ячейки учитываются. Например 'age'>4 равносильно 'age' > 4.
12. Наименование колонок и строковые значения оборачиваются в одинарные кавычки. Например наименование колонки age в запросе должна быть в одинарных кавычках (как в примерах запроса). А числа и булевые значения без кавычек (как в примерах запроса).
13. Значения которые передаются на сравнение не могут быть null. А при записи значение колонки может быть null. Т.е. 'age'>=null считаем, что такого не может быть. А UPDATE VALUES 'age'=null - может быть, в этом случае значение из ячейки удаляется.
14. Если значение в ячейке например age пусто (null), а на вход передается условие типа 'age'!=0 , а существующее значение age=null. То запрос считается корректным, 0!=null.

15. Колонки и команды должны быть регистро независимыми. Т.е. INSERT тоже самое, что insert. lastName тоже самое, что и lastname и LASTNAME. WHERE тоже самое, что и where.
16. Код должен быть компилируемым и исполняем под OpenJDK 18.
17. Сторонние библиотеки использовать нельзя. Реализация на чистом OpenJDK 18.
18. Данные в коллекции должны сохраняться на время исполнения программы. Т.е. последовательно можно вызвать несколько команд. Как в примере ниже.

Пример вызова:

```
package test;


import java.util.List;
import java.util.Map;


public class Main {


   public static void main(String... args){
        Test starter = new Test ();
        try {
            //Вставка строки в коллекцию
            List<Map<String,Object>> result1 = starter.execute("INSERT VALUES 'lastName' = 'Федоров' , 'id'=3, 'age'=40, 'active'=true");
            //Изменение значения которое выше записывали
            List<Map<String,Object>> result2 = starter.execute("UPDATE VALUES 'active'=false, 'cost'=10.1 where 'id'=3");
            //Получение всех данных из коллекции (т.е. в данном примере вернется 1 запись)
            List<Map<String,Object>> result3 = starter.execute("SELECT");


        }catch (Exception ex){
            ex.printStackTrace();
        }
   }
}
```