Assignment 3 report 1

Team 2:

108020020 劉俊鍵

108020039 陳裕翔

1. Implementation

(1) First, in the package `org.vanilladb.core.query.parse.Lexer`
We need to add “explain” to keywords in the initKeywords() to make sure that lexer recognize “explain” command.

(2) Secondly, in the package `org.vanilladb.core.query.parse.Parser`
In order to parse query command correctly, we need to modify queryCommand() method in the Parser class, since the query command is whether with “explain” or without “explain”, here we use a Boolean type, isExplained, to record whether there is a “explain” or not. The default value of isExplained will set to false. In the queryCommand() method, we will first to match the “explain” keywords. If matched the lex.eatKeyword(“explain”) will be called, then isExplained will be set to true, and continuously parse remaining query command. And Finally we call QueryData() constructor to constructor querydata, which will pass the argument isExplained.

(3) Then, in the package `org.vanilladb.core.query.parse.QueryData`
We need to add a private class member, isExplained, which is Boolean type, isExplained is a record whether the query command have “explain” or not.
So, in the constructor, we need to set isExplained through the argument passed from Parser.

(4) in the `org.vanilladb.core.query.planner.BasicQueryPlanner`
The heuristic query planning algorithm suggest that explain must in the top level, so we deal with explain in the last, that is step 7 in the createPlan() method in the BasicQueryPlanner.

(5) In the `org.vanilladb.core.query.algebra.Plan`
We add a method recordData() : String, which will record the planning string in the plan, such as TablePlan, ProductPlan, SelectPlan, ProjectPlan, GroupByPlan, SortPlan, and so on. So, recordData() will implemented by each Plan to record the current planning string.

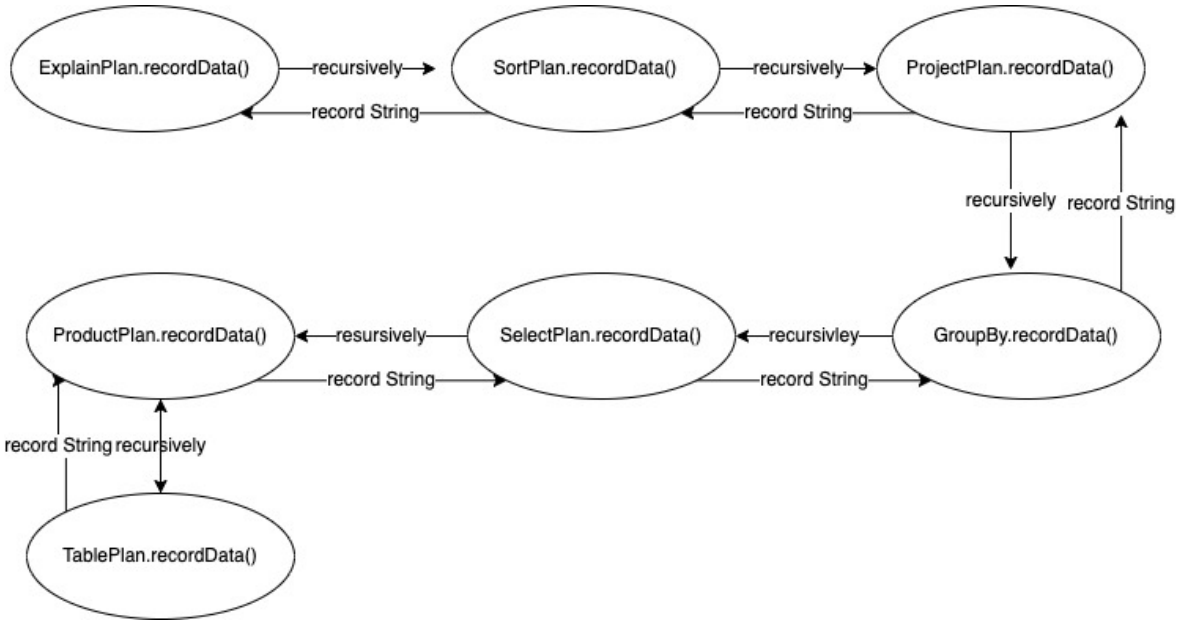
(6) In the `org.vanilladb.core.query.algebra.ExplainPlan`
We add a class ExplainPlan in the `org.vanilladb.core.query.algebra.`
In the ExplainPlan, we need a schema, which have a field called “query-plan” with type equal to Type.VARCHAR(500). When scheman() is called we will return schema so that the program will not crash. If open() is called we need to calculate the planning string. So we called recordData() recursively to record the planning string, and recordData() will return a String, which is a planning string we want . Then we pass p.open() and recordData() to ExplainScan constructor.

Note that, each plan, such as TablePlan, ProductPlan, SelectPlan, ProjectPlan, GroupByPlan, SortPlan will implement recordData() to record planning string recursively down to TablePlan.

(7) In the `org.vanilladb.core.query.algebra.ExplainScan`

Since ExplainScan only has one record, we constructor a private class member isBeforeFirst to record whether is in the beforefirst or not. If beforeFirst() is called we set isBeforeFirst to true. If next() is called, if isBeforeFirst is true we return true, and set isBeforeFirst ot false; if isBeforeFirst is false false we return false, since ExplainScan only have one record, so a Boolean isBeforeFirst to represent is enough. And here we construct a actualRun() method to run the command acutally and record the number of record. Finally, Once getVal() is called we return VarcharConstant(explainRecord), which is defined in sql type.

The (6),(7) can be showed as following graph.



2. Explain Result

(1) A query accessing single table with WHERE

SQL > EXPLAIN SELECT d_id FROM district WHERE d_w_id < 5

query-plan

->ProjectPlan (#blks=2, #recs=10)

->SelectPlan pred:(d_w_id<5.0(#blks=2, #recs=10)

->TablePlan on(district)(#blks=2, #recs=10)

Actual #recs: 10

(2)A query accessing multiple tables with WHERE

```
SQL> EXPLAIN SELECT d_id FROM district, warehouse WHERE d_w_id = w_id

query-plan
-------------------------------------------------------------------------------
->ProjectPlan    (#blks=22, #recs=10)
    ->SelectPlan    pred:(d_w_id=w_id(#blks=22, #recs=10)
        ->ProductPlan    (#blks=22, #recs=10)
            ->TablePlan on(district)(#blks=2, #recs=10)
            ->TablePlan on(warehouse)(#blks=2, #recs=1)
Actual #recs: 10
```

(3)A query with ORDER BY

```
SQL> EXPLAIN SELECT d_id,d_name FROM district ORDER BY d_id DESC

query-plan
-------------------------------------------------------------------------------
->SortPlan    (#blks=1, #recs=10)
    ->ProjectPlan    (#blks=2, #recs=10)
        ->SelectPlan    pred:((#blks=2, #recs=10)
            ->TablePlan on(district)(#blks=2, #recs=10)
Actual #recs: 10
```

(4)A query with GROUP BY and at least one aggregation function (MIN, MAX, COUNT, AVG... etc.)

```
SQL> EXPLAIN SELECT COUNT(d_id) FROM district GROUP BY d_city

query-plan
-------------------------------------------------------------------------------
->ProjectPlan    (#blks=1, #recs=10)
    ->GroupByPlan    (#blks=1, #recs=10)
        ->SortPlan    (#blks=1, #recs=10)
            ->SelectPlan    pred:((#blks=2, #recs=10)
                ->TablePlan on(district)(#blks=2, #recs=10)
Actual #recs: 10
```