team2_assignment3_report1

members:

- 109062233 蘇裕恆
- 109062320 朱季葳
- 109062143 林聖哲

Implementation

Parse

LEXER

We add `explain` to the keywords array.

PARSER

1. We set a flag call `explain_flag` , and we rise the flag when `lex.matchKeyword("explain")` is true.Then we eat the keyword, which is just the same as the implementation of all the other command.
2. We changed the format of the constructor of QueryData, and rewrite the return function's format to meet the one of QueryData, that is, add `explain_flag` 's part.

QUERYDATA

1. We add a member variable called `explainFlag` to QueryData class, and its usage is the same as what we've mentioned in the parser part.
2. In the `toString()` function, we append "explain" to the result when `explainFlag` is risen.

Planner

BASICQUERYPLANNER

We add a new if statement in the end to let the `createPlan` function can create a new `ExplainPlan` object when the *explainFlag* in data is set to true, which ment that there is a explainplain in the top of the plan tree.

Algebra

Plan

Since we need to call the recurse function to output the result of Explain, so we add a `toString` function to handle this function.

ExplainPlan

Since the property of "explain" is similar to "project", we take ProjectPlan as reference here.
Besides, since **an output of Explain should be a record containing a value in the field query-plan**, we add a field called "query-plan" into the schema.
As for the `toString()` part, we use a for loop to count the total records, and append the result of recursive call on `toString()`, and the amount of total record as the format shown in spec.Finally, we return the string we just made.

```
public String toString(){
    StringBuilder result = new StringBuilder();
    Scan s = p.open();
    int rec_num = 0;
    s.beforeFirst();
    while(s.next())rec_num++;
    s.close();
    result.append("\n" + p.toString() + "Actual #recs: " + rec_num);
    return result.toString();
}
```

ExplainScan

Just the same as ExplainPlan part, we take ProjectScan as reference here.
The different part is that we have the output string and explain flag, and we only have one `Scan` object here, so we have the following difference.

1. Constructor

```java
public ExplainScan(Scan s,String ans) {
    this.s = s;
    this.ans = ans;
    this.isExplain = false;
}
```

2. member variable
    - Scan s
    - String ans
    - Boolean explain

3. `next()` :Since all the results will only stored in `query-plan` field, and if `isExplain` is true, it will be the only one member in the ResultSet, we return false if `isExplain` is true.

4. `getVal()` :Since we only have a field called “query-plan”, we return false if `fidname != "query-plan"`

5. `hasField(String fldName)` : return true if this s has the corresponding field.

OTHER PLANS

Implement the the overide `toString` function to fit the output of the Explain mode.

EXPLAIN result

- Use our `bench` project to load the TPC-C testbed and show the `EXPLAIN` result for the following queries:
    - A query accessing single table with `WHERE`
    - A query accessing multiple tables with `WHERE`
    - A query with `ORDER BY`
    - A query with `GROUP BY` and at least one aggregation function (`MIN`, `MAX`, `COUNT`, `AVG`... etc.)

A query accessing single table with WHERE

EXPLAIN SELECT d_id from district where d_w_id > 5

```
SQL> EXPLAIN SELECT d_id FROM district WHERE d_w_id > 5

query-plan
------------------------------------------------------------
->ProjectPlan  (#blks=2, #recs=0)
        ->SelectPlan pred:(d_w_id>5.0) (#blks=2, #recs=0)
                ->TablePlan on (district) (#blks=2, #recs=10)
Actual #recs: 0
```

A query accessing multiple tables with WHERE

EXPLAIN SELECT w_street_1 FROM warehouse , district WHERE w_zip = d_zip;

```
SQL> EXPLAIN SELECT w_street_1 FROM warehouse , district WHERE w_zip = d_zip;

query-plan
-----------------------------------------------------------------------------------------
->ProjectPlan  (#blks=22, #recs=0)
        ->SelectPlan pred:(w_zip=d_zip) (#blks=22, #recs=0)
                ->ProductPlan  (#blks=22, #recs=10)
                        ->TablePlan on (district) (#blks=2, #recs=10)
                        ->TablePlan on (warehouse) (#blks=2, #recs=1)
Actual #recs: 0
```

A query with ORDER BY

EXPLAIN select d_id, d_name from district where d_id < 5 order by d_name DESC;

```
SQL> EXPLAIN select d_id, d_name from district where d_id < 5 order by d_name DESC;

query-plan
-----------------------------------------------------------------------------------------
->SortPlan (#blks=0, #recs=0)
        ->ProjectPlan  (#blks=2, #recs=0)
                ->SelectPlan pred:(d_id<5.0) (#blks=2, #recs=0)
                        ->TablePlan on (district) (#blks=2, #recs=10)
Actual #recs: 4
```

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

EXPLAIN SELECT COUNT(d_id) FROM district, warehouse WHERE d_w_id = w_id GROUP BY w_id

```
SQL> EXPLAIN SELECT COUNT(d_id) FROM district, warehouse WHERE d_w_id = w_id GROUP BY w_id

query-plan
-----------------------------------------------------------------------------------------------
->ProjectPlan  (#blks=2, #recs=1)
        ->GroupByPlan: (#blks=2, #recs=1)
                ->SortPlan (#blks=2, #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: 1
```