Introduction to Database Systems

Assignment 3 Phase 1 report
Team 8: 107060025 王子文、107062312 郭濬睿

Implementation

Parser and Lexer

First, add "explain" as a keyword in Lexer, and when Parser read input query, determine whether there's an "explain" keyword.

QueryData

Add a private variable "isExplain" to store if this query data is an "explain" query.

BasicQueryPlanner

In query planner, check if input query data is an "explain" query. If yes, wrap current plan in an "explain plan".

ExplainPlan/ExplainScan and other Plans

We create two new classes to deal with "explain" query.

In ExplainPlan, we take a lower level plan as input, construct a schema that only contains a "query-plan" field, then prepare the desired output message. The way we generate this message is, we add a method "toString" in other type of plans, this method will return a string that contain the message of that plan. By calling "toString" recursively, we can have the entire message in ExplainPlan. Also, to get the number of records actually accessed, we open the plan to get a scan, then use a while loop to count the number of times that next() can be called in this scan.

When calling plan.open(), ExplainPlan will pass the prepared message into a new ExplainScan and return it. In ExplainScan, when calling getVal(), it will check if the field is "query-plan" and return the prepared message. Since it should contain only one record, we use a boolean variable(*BV*) in ExplainScan to check: when beforeFirst() is called, set *BV* to true; when next() is called, if *BV* is true, set *BV* to false and return true, otherwise return false.

Experiment

- A query accessing single table with WHERE

```
EXPLAIN SELECT c_id, c_first FROM customer WHERE c_id < 100
```

```
query-plan
------------------------------------------------------------
->ProjectPlan (#blks=15001, #recs=1030)
  ->SelectPlan pred:(c_id<100.0) (#blks=15001, #recs=1030)
    ->TablePlan on (customer) (#blks=15001, #recs=30000)
Actual #recs: 990
```

- A query accessing multiple tables with WHERE

```
EXPLAIN SELECT d_id, no_o_id FROM district, new_order WHERE d_id = no_d_id
```

```
query-plan
------------------------------------------------------------
->ProjectPlan (#blks=372, #recs=4764)
  ->SelectPlan pred:(d_id=no_d_id) (#blks=372, #recs=4764)
    ->ProductPlan (#blks=372, #recs=90000)
      ->TablePlan on (district) (#blks=2, #recs=10)
      ->TablePlan on (new_order) (#blks=37, #recs=9000)
Actual #recs: 9000
```

- A query with ORDER BY

```
EXPLAIN SELECT h_c_id, h_date FROM history ORDER BY h_date DESC
```

```
query-plan
------------------------------------------------------------
->SortPlan (#blks=118, #recs=30000)
  ->ProjectPlan (#blks=859, #recs=30000)
    ->SelectPlan pred:() (#blks=859, #recs=30000)
      ->TablePlan on (history) (#blks=859, #recs=30000)
Actual #recs: 30000
```

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

```
EXPLAIN SELECT MAX(o_ol_cnt) FROM orders GROUP BY o_c_id
```

```
query-plan
------------------------------------------------------
->ProjectPlan (#blks=295, #recs=1087)
  ->GroupByPlan (#blks=295, #recs=1087)
    ->SortPlan (#blks=295, #recs=30000)
      ->SelectPlan pred:() (#blks=296, #recs=30000)
        ->TablePlan on (orders) (#blks=296, #recs=30000)
Actual #recs: 3000
```

- Other thinig worth mentioning
  - When accessing multiple tables, it will be extremely slow when two table size are large even if there are `WHERE` constrain.