Database Assignment 3 Report 1

TEAM 6 107062318 李俊逸 107062202 陳敬和 107062237 張濬洋

A. How We Implement The EXPLAIN Operation

1. Lexer, Parser

i. Lexer

We add a keyword “explain” in Collection<String> keywords.

```
private void initKeywords() {
    keywords = Arrays.asList("select", "from", "where", "and", "insert",
            "into", "values", "delete", "drop", "update", "set", "create", "table",
            "int", "double", "varchar", "view", "as", "index", "on",
            "long", "order", "by", "asc", "desc", "sum", "count", "avg",
            "min", "max", "distinct", "group", "add", "sub", "mul", "div",
            "using", "hash", "btree", "explain");
}
```

ii. Parser

We modified Parser queryCommand() to get the keyword “explain” by matching and eating function. Also, we modified the return function to verify whether it is an explain query.

```
public QueryData queryCommand() {
    boolean explain = false;
    if (lex.matchKeyword("explain")) {
        lex.eatKeyword("explain");
        explain = true;
    }
```

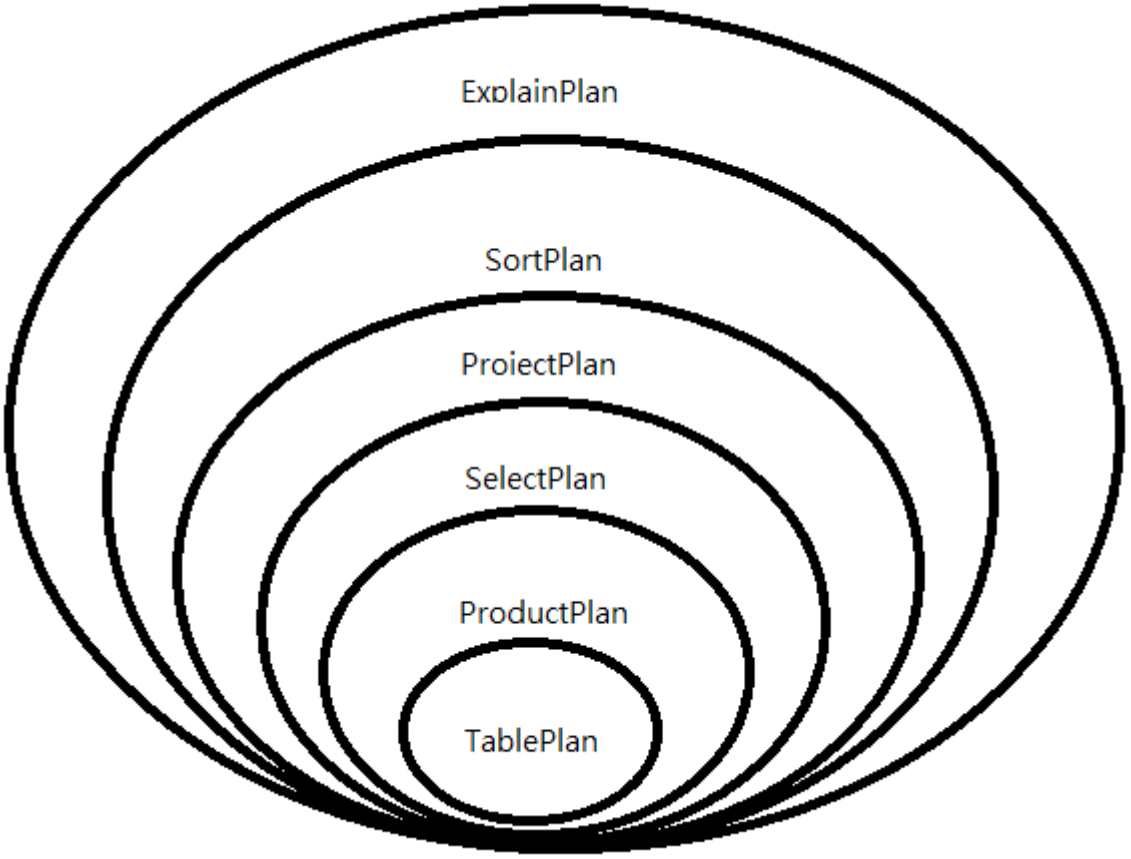
```
return new QueryData(projs.asStringSet(), tables, pred,
        groupFields, projs.aggregationFns(), sortFields, sortDirs, explain);
```

In QueryData, we add a new boolean variable ‘explain’ to record if the query is Explain or not. To meet the requirement above, we add the explain argument to the QueryData constructor.

```
public QueryData(Set<String> projFields, Set<String> tables, Predicate pred,
        Set<String> groupFields, Set<AggregationFn> aggFn, List<String> sortFields, List<Integer> sortDirs, boolean explain) {
    this.explain = explain;
    this.projFields = projFields;
    this.tables = tables;
    this.pred = pred;
    this.groupFields = groupFields;
    this.aggFn = aggFn;
    this.sortFields = sortFields;
    this.sortDirs = sortDirs;
}
```

We also add a new boolean function isExplainPlan() to get the value of “explain”.

2. Hierarchy structure of Plans and their .toString() functions



Inside each plan, we implemented a toString function. Take ProductPlan as an example:

```
public String toString() {
    String T1 = p1.toString();
    String T2 = p2.toString();
    StringBuilder s = new StringBuilder();
    s.append("->ProductPlan  (#blks=").append(blocksAccessed()).append(", #recs=").append(recordsOutput()).append(")\n");
    s.append("\t").append(T1).append("\t").append(T2);
    return s.toString();
}
```

It returns the string which contains the query plan we want with the above hierarchy.

3. ExplainPlan, ExplainScan

To implement a plan, we need at least 5 functions, which are open(), blocksAccessed(), schema(), histogram(), and recordsOutput(). For the blocksAccessed(), histogram(), we just output the output of the previous plan. For the open(), we create an ExplainScan() according to the previous scan, schema, and the explain(i.e. toString()), which is the EXPLAIN output of the previous plan.For the schema(), we simply add a field called “query-plan”. For the recordsOutput(), we simply set its output as 1.

The next one is to implement ExplainScan. As a scan, we need to implement 5 functions, which are constructor(), beforeFirst(), next(), close(), hasField(). The inputs of ExplainScan are schema, scan from the previous plan, and the result of EXPLAIN from the previous plan. We need an EXPLAIN operation to tell us the amount of records and the whole plan tree. So in the constructor, we calculate the amount of records from the input scan by calling its next(). Also, we produce the tree in terms of String, by calling the previous

plan tree toString() function. For the beforeFirst() and next(), since we only output once for each time it is opened, we set a boolean value to represent its state. If it is opened and not accessed, return the EXPLAIN result and set the value as false. If it is accessed, we cannot access it again until it is reopened. For the getVal(), if the input field is “query-plan”, then we will output its value.

B. SQL EXPLAIN Query Results

1. A query accessing single table with WHERE

EXPLAIN SELECT c_street_1 FROM customer WHERE c_id=1;

```
SQL> EXPLAIN SELECT c_street_1 FROM customer WHERE c_id=1;

query-plan
-----------------------------------------------------------------------------
->ProjectPlan: (#blks=15001, #recs=28)
        ->SelectPlan pred:(c_id=1.0) (#blks=15001, #recs=28)
                ->TablePlan on (customer) #blks=15001, #recs=30000)

 Actual #recs: 10
```

2. A query accessing multiple tables with WHERE

EXPLAIN SELECT d_id FROM district, warehouse WHERE d_w_id = w_id;

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

query-plan
-----------------------------------------------------------------------------
->ProjectPlan: (#blks=43, #recs=40)
        ->SelectPlan pred:(d_w_id=w_id) (#blks=43, #recs=40)
                ->ProductPlan  (#blks=43, #recs=40)
                        ->TablePlan on (district) #blks=3, #recs=20)
                        ->TablePlan on (warehouse) #blks=2, #recs=2)

 Actual #recs: 40
```

3. A query with ORDER BY

EXPLAIN SELECT c_id, c_first FROM customer ORDER BY c_first;

```
SQL> EXPLAIN SELECT c_id, c_first FROM customer ORDER BY c_first;

query-plan
-------------------------------------------------------------------------------

        ->ProjectPlan: (#blks=15001, #recs=30000)
                ->SelectPlan pred:() (#blks=15001, #recs=30000)
                        ->TablePlan on (customer) #blks=15001, #recs=30000)

 Actual #recs: 30000
```

4. A query with GROUP BY and at least one aggregation function (MIN, MAX, COUNT, AVG... etc.)
EXPLAIN SELECT MAX(d_id) FROM district GROUP BY d_name;

```
SQL> EXPLAIN SELECT MAX(d_id) FROM district GROUP BY d_name;

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

SQL>
```