Cloud Databases Assignment 2

Group 2
102061105 Yun-Sheng, Chang
102061212 Jian-Hao, Huang

April 5, 2017

1 Implementation

1.1 New interface Explainable

To support the *EXPLAIN* operation, we first define a new interface *Explainable*,

```
1 public interface Explainable {
2   String getStatisticResult(int level);
3 }
```

Explainable.java

and have every plans that would possibly be created by the *BasicQueryPlanner* implement the interface.

```
1 public class TablePlan implements Plan, Explainable
2 public class ProductPlan implements Plan, Explainable
3 public class SelectPlan extends ReduceRecordsPlan implements Explainable
4 public class GroupByPlan extends ReduceRecordsPlan implements Explainable
5 public class ProjectPlan implements Plan, Explainable
6 public class SortPlan implements Plan, Explainable
```

Plans implementing Explainable

Three things should be done in *getStatisticResult()*:

1. indentation;
2. form the statistic result of the current plan;
3. call p.getStatisticResult() to obtain the statistic results of the underlying plans.

We show the implementation in *SelectPlan* as an example:

```
1  public String getStatisticResult(int level) {
2    Explainable p;
3    String res = new String();
4
5    // step 1: indentation
6    for (int i = 0; i < level; i++)
7      res += "\t";
8
9    // step 2: form the statistic result of the current plan
10   res += "->SelectPlan pred:(" + this.pred.toString() + ") " +
11     "(#blks=" + blocksAccessed() + ", #recs=" + recordsOutput() + ")\n";
12
13   // step 3: call p.getStatisticResult() to obtain the statistic result of the underlying plans
14   p = (Explainable)this.p;
15   res += p.getStatisticResult(level + 1);
16
17   return res;
18 }
```

SelectPlan.java

The argument *level* represents the level of the current plan in the plan tree. For instance,

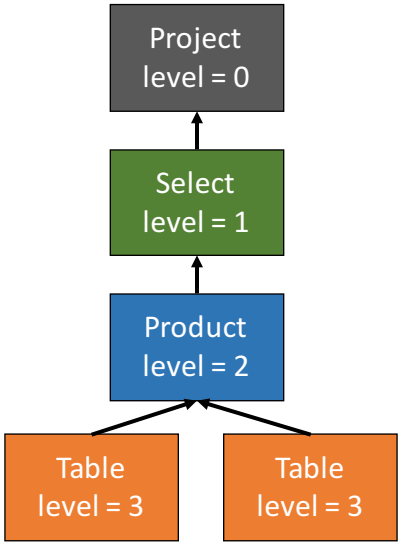

Figure 1: Level of plans

With the value of *level*, we will be able to do the indentation properly.

```
// step 1: indentation
for (int i = 0; i < level; i++)
  res += "\t";
```

Indentation

Then, we can obtain the estimated number of accessed blocks and records with *blocksAccessed()* and *recordsOutput()*, respectively.

```
// step 2: form the statistic result of the current plan
res += "->SelectPlan pred:(" + this.pred.toString() + ") " +
  "(#blks=" + blocksAccessed() + ", #recs=" + recordsOutput() + ")\n";
```

Form statistic result of the current plan

We call *p.getStatisticResult()* to get the results of the underlying plans.

```
// step 3: call p.getStatisticResult() to obtain the statistic result of the underlying plans
p = (Explainable)this.p;
res += p.getStatisticResult(level + 1);

return res;
```

Obtain statistic results of the underlying plans

Finally, we return the result to the upper plan.

1.2 Lexer, Parser, QueryData and BasicQueryPlanner

The keyword *explain* is added to the keyword list.

```
private void initKeywords() {
  keywords = Arrays.asList("select", "from", "where", "and", "insert",
      ...
      "min", "max", "distinct", "group", "add", "sub", "mul", "div", "explain");
}
```

Lexer.java

We modify the parsing procedure to check if the query is an *EXPLAIN* query; if so, set the flag *isExplain*.

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

  lex.eatKeyword("select");
  ProjectList projs = projectList();
  // parse other tokens

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

Parser.java

When the query is successfully parsed, the query's information will be stored in the data structure *QueryData* and passed to the *BasicQueryPlanner*. Thus, we need to add an attribute *isExplain* to the *QueryData*.

```
private boolean isExplain;

public boolean isExplain() {
  return isExplain;
}
```

QueryData.java

Finally, the *BasicQueryPlanner* should create the *ExplainPlan* as the top-level plan for an *EXPLAIN* query.

```
public Plan createPlan(QueryData data, Transaction tx) {
  // create other plans

  if (data.isExplain())
    p = new ExplainPlan(p);
  return p;
}
```

BasicQueryPlanner.java

1.3 ExplainPlan

The result of an *EXPLAIN* query contains one column, *query-plan*; we add the column to the schema in the constructor of *ExplainPlan*.

```
public ExplainPlan(Plan p) {
  this.p = p;
  this.hist = p.histogram();
  schema.addField("query-plan", Type.VARCHAR(500));
}
```

ExplainPlan.java

Before the *ExplainScan* is created, we must collect the statistic result of other plans by invoking the method we previously implement: *getStatisticResult()*.

```
public Scan open() {
  String str;

  str = ((Explainable)p).getStatisticResult(0);
  Scan s = p.open();
  return new ExplainScan(s, str);
}
```

ExplainPlan.java

1.4 ExplainScan

ExplainScan has the following attributes:

```
1  private Scan s;
2  private String str;
3  private boolean first;
4  private int numOfRec;
5  private SpResultRecord rec;
```

ExplainScan.java

s is the scan of the underlying query.
str is the statistic results.
first is true if the client has called *beforeFirst()* and has not called *next()* yet.
numOfRec is the actual number of records of the *SELECT* query.
rec is the table storing the result of the *EXPLAIN* query.

In the constructor of ExplainScan, we first count the actual number of records by putting *s.next()* in a while loop and increase *numOfRec* by one every time the condition holds. Then, we append the actual number of records to the statistic result and add the final result to the table.

```
1  public ExplainScan(Scan s, String str) {
2    s.beforeFirst();
3    while (s.next())
4      numOfRec++;
5
6    rec.setVal("query-plan",
7        new VarcharConstant("\n" + str + "\nActual #recs: " +
8        numOfRec, Type.VARCHAR(500)));
9  }
```

ExplainScan.java

Since the result of the *EXPLAIN* query contains exactly one record, we set the flag *first* when *beforeFirst()* is invoked and reset it when *next()* is invoked.

```
1  public void beforeFirst() {
2    s.beforeFirst();
3    first = true;
4  }
5
6  public boolean next() {
7    if(first == true){
8      first = false;
9      return true;
10   }
11   return false;
12 }
```

ExplainScan.java

2 Results

2.1 Single table A query with WHERE

```
SQL> SELECT d_id FROM district WHERE d_w_id = 2

  d_id
------
     1
     2
     3
     4
     5
     6
     7
     8
     9
    10

SQL> EXPLAIN SELECT d_id FROM district WHERE d_w_id = 2

query-plan
-----------------------------------------------------------------
->ProjectPlan  (#blks=3, #recs=10)
        ->SelectPlan pred:(d_w_id=2.0) (#blks=3, #recs=10)
                ->TablePlan on (district) (#blks=3, #recs=20)

Actual #recs: 10
```

2.2 Multiple tables query with WHERE

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

  d_id
------
     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
     1
     2
     3
     4
     5
     6
     7
     8
     9
    10

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

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

Actual #recs: 20
```

2.3 Query with ORDER BY

```
SQL> SELECT d_id FROM district, warehouse WHERE d_w_id = w_id ORDER BY d_id ASC

  d_id
------
     1
     1
     2
     2
     3
     3
     4
     4
     5
     5
     6
     6
     7
     7
     8
     8
     9
     9
    10
    10

SQL> EXPLAIN SELECT d_id FROM district, warehouse WHERE d_w_id = w_id ORDER BY d_id ASC

query-plan
-------------------------------------------------------------------------------------------
->SortPlan  (#blks=1, #recs=20)
        ->ProjectPlan  (#blks=43, #recs=20)
                ->SelectPlan pred:(d_w_id=w_id) (#blks=43, #recs=20)
                        ->ProductPlan  (#blks=43, #recs=40)
                                ->TablePlan on (district) (#blks=3, #recs=20)
                                ->TablePlan on (warehouse) (#blks=2, #recs=2)

Actual #recs: 20
```

2.4 Query with GROUP BY and at least one aggregation function

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

count_of_d_id
------
    10
    10

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

query-plan
-----------------------------------------------------------------------------------------------
->ProjectPlan  (#blks=4, #recs=2)
        ->GroupByPlan  (#blks=4, #recs=2)
                ->SortPlan  (#blks=4, #recs=20)
                        ->SelectPlan pred:(d_w_id=w_id) (#blks=43, #recs=20)
                                ->ProductPlan  (#blks=43, #recs=40)
                                        ->TablePlan on (district) (#blks=3, #recs=20)
                                        ->TablePlan on (warehouse) (#blks=2, #recs=2)

Actual #recs: 2
```