108062182劉家成 108062281莊晴雯 106000202 洪心苹

<u>How We Implemented Explain Operation</u>

We created two files for the EXPLAIN implementation:

- ExplainPlan.java
- ExplainScan.java

<u>ExplainPlan.java</u>

In the class ExplainPlan, we first have the constructor ExplainPlan that takes the dummy parameter p with type Plan, and sets this.p = p.

Next, we created a project scan for this query which will call p.open() and store it to the variable s with the type Scan, then return a new object by calling the class constructor ExplainScan(s, toString(), schema()).

We also estimated the number of block accesses in the projection, which can be implemented by simply returning p.blockAccessed() in the blockAccessed() function, since the value is the same as the one in the underlying query.

In the function schema(), which is used to return the schema of the projection taken from the field list, we created a new object Schema which we stored inside the explainSchema variable which has the type Schema, then use the addField method so that a record containing a value in the field query-plan with the type VARCHAR(500) can be added later on. Lastly we simply need to return the explainSchema at the end of this function.

For the histogram function, we simply need to return p.histogram() which holds the histogram that approximates the joint distribution of the field values of query results.

Since we know that the estimate of the number of records in the query’s output table is 1, we simply return 1 in the recordsOutput() function.

Lastly, for the toString() funcion, we returned the variable c with the type String, which holds the value of the return value of p.toString().

<u>ExplainScan.java</u>

Constructor

We have the ExplainScan constructor which sets the following values:

- this.explainRes = “\n” + explainRes
- this.schema = schema

The explainRes mentioned above holds the value of the plan tree with the estimated number of blocks and records for each plan. After setting the above two values, we can call s.beforeFirst() which sets the flag isFirst to true for the initial action, then we used a while loop to check for s.next() and increments the value of totalRecs while s.next() is not empty. After we finish iterating through s.next(), we can call s.close() to finish the action. Afterwards, we can append the value of totalRecs to the value of explainRes, which can be implemented in the following way:

- this.explainRes = this.explainRes + “\nActual #recs: “ + this.totalRecs

Then since we have finished everything we needed to do, we set the isFirst value back to true since we finished this execution, and so that the next execution can run.

- this.isFirst = true

beforeFirst()
Sets the isFirst flag to true to indicate that the execution just began.

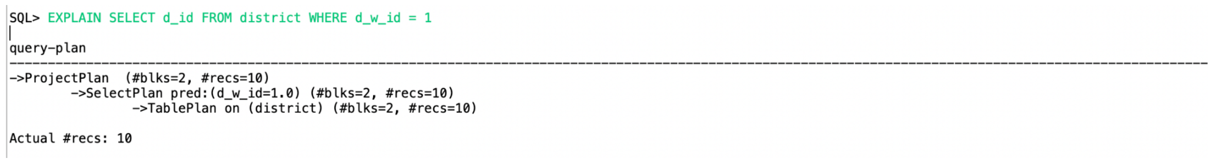
next()
Toggles the isFirst value of isFirst == true and returns true, but directly returns isFirst if isFirst's value is false. This method will help in iterating through s to count the total records by setting the counted records to false.

getVal()
The two conditions in this function were when the field name is equal to "query-plan" or when it is not. When it is equal to the string "query-plan", then we can return a new VarcharConstant object and pass this.explainRes to the constructor. Else, we will throw a Runtime Exception and state that the field name is not found.

hasField()
Simple returns schema.hasField(fldName).

__**EXPLAIN result for the following queries:**__

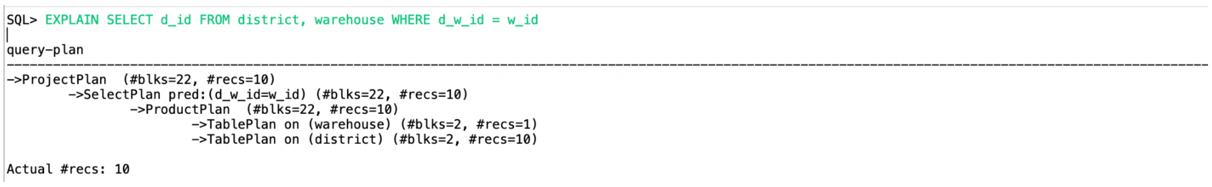
● A query accessing single table with WHERE



```
SQL> EXPLAIN SELECT d_id FROM district WHERE d_w_id = 1
|
query-plan
----------------------------------------------------------------------------------------------------------------------------------------
->ProjectPlan  (#blks=2, #recs=10)
        ->SelectPlan pred:(d_w_id=1.0) (#blks=2, #recs=10)
                ->TablePlan on (district) (#blks=2, #recs=10)

Actual #recs: 10
```

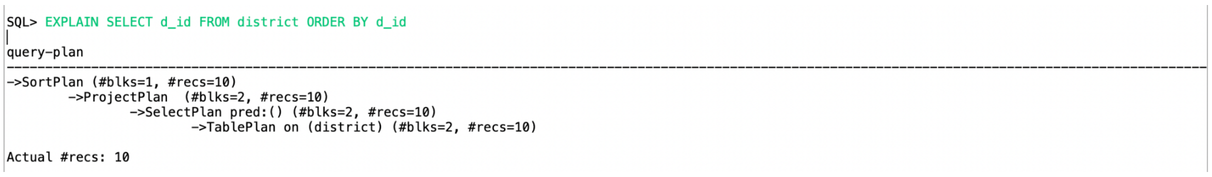
● A query accessing multiple tables with WHERE



```
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 (warehouse) (#blks=2, #recs=1)
                        ->TablePlan on (district) (#blks=2, #recs=10)

Actual #recs: 10
```

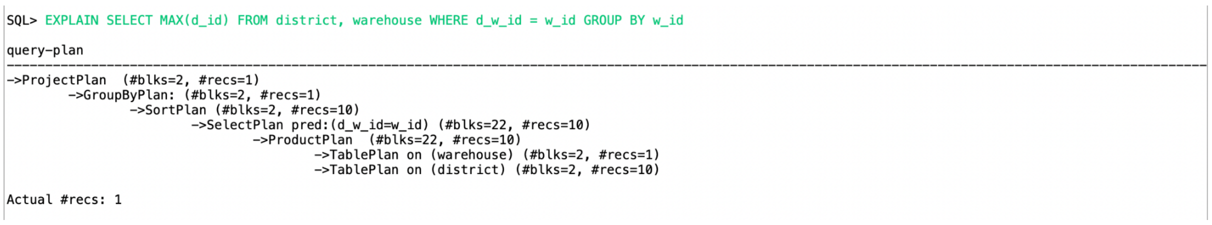
● A query with ORDER BY



```
SQL> EXPLAIN SELECT d_id FROM district ORDER BY d_id
|
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
```

● A query with GROUP BY and at least one aggregation function ()



```
SQL> EXPLAIN SELECT MAX(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 (warehouse) (#blks=2, #recs=1)
                                        ->TablePlan on (district) (#blks=2, #recs=10)

Actual #recs: 1
```

Anything Worth To Be Mentioned

In the ProductPlan, ProjectPlan, and SelectPlan classes, we added a toString() function to return a string which contains the information about ProductPlan/ProjectPlan/SelectPlan including the number of blocks accessed and the records number. To implement this, we first convert p to string and store it in c with the type String, then split c whenever “\n” is encountered. We do the same steps if we have more than 1 p. Then we created a new StringBuilder object called sb and appended the information about the number of blocks which can be obtained by using the blockAccessed() method and also the number of records by using the recordsOutput() method. We arrange the string with the format: ->{Product/Project/Select}Plan (#blks=..., #recs=...). We can then append the values of child along with one tab and newline, and return sb which is converted into string.

In ExplainScan we use the flag isFirst because if we don’t set the value of next() to false on the second call then it will re-run the explain plan which is not the behavior we wanted, hence we used a variable to tell us if it has run once or not.