Assignment 3 Phase 1 Report

- Team10
  - 108060031 黃瑋辰
  - 108060030 陳昱穎
  - 108062308 黃俊瑋

Implementing “EXPLAIN” SQL command

Lexer, Parser & QueryParser

The classes we modified or added:

- `org.vanilladb.core.query.parse.lexer`
- `org.vanilladb.core.query.parse.parser`
- `org.vanilladb.core.query.parse.QueryData`

Modifying the two classes to support decompose the commands including “EXPLAIN” keyword. Modifying the `QueryData` to support “EXPLAIN” plan and schema.

Lexer

File: `org.vanilladb.core.query.parse.lexer`

In this class, we add the keyword “explain” to `keywords` list in function `initKeyword()`.

Parser

File: `org.vanilladb.core.query.parse.parser`

In parser class, to reuse the function of `queryCommand()`, we just add some condition check to determine whether the query string containing “EXPLAIN” or not. If contains, we set the new variable `isExplain` as `true`. Then, pass the variable into the `QeuryData` with modification of its constructor.

QueryParser

File: `org.vanilladb.core.query.parse.QueryData`

In this class, for the purpose of supporting the later function in other package (`BasicQueryPlanner`), we add a private variable `isExplain` and modify its constructor function.

Planner

The classes we modified or added:

- `org.vanilladb.core.query.planner.BasicQueryPlanner`

Modifying these packages to support creating explain plan return value.

BasicQueryPlanner

File: `org.vanilladb.core.query.planner.BasicQueryPlanner`

Inside this class implementation, we only add a condition check: if the `data.isExplain` is true, it means the query is an EXPLAIN command. So that, we let `p = new ExplainPlan(p);`.

Plan

The classes we modified or added:

- `org.vanilladb.core.query.algebra.TablePlan`
- `org.vanilladb.core.query.algebra.SelectPlan`
- `org.vanilladb.core.query.algebra.ProductPlan`
- `org.vanilladb.core.query.algebra.ProjectPlan`
- `org.vanilladb.core.query.algebra.ExplainPlan`

For existing plan, we just modified there `toString()` function for them to return the query plan we want.

ExplainPlan

File: `org.vanilladb.core.query.algebra.ExplainPlan`

In this file, we mimic this class as other plan. However, we add a field named “query-plan” into its schema for later use in `ExplainScan`. For `toStirng()`, the actual records is defined as the last plan’s `recordsOutput()`.

In the `open()`, we called the subscan `s.beforeFirst()`. Then iterate `s.next()` to confirm how many actual records are.

ExplainScan

The classes we modified or added:

- `org.vanilladb.core.query.algebra.ExplainScan`

In this scan, it is quite like other scans. But we adjust the function `getval()`. If the field name is “query-name”, the function will return the query plan string back, which will be access by `RemoteStatementImpl` & `ConsoleSQLInterpreter`.

Benchmarking

- A query accessing single table with `WHERE`

```
SQL> EXPLAIN SELECT s_i_id FROM stock WHERE s_i_id <= 3

query-plan
-----------------------------------------------------------------
->ProjectPlan  (#blks=25001, #recs=0)
        ->SelectPlan pred:(s_i_id<=3.0) (#blks=25001, #recs=0)
                ->TablePlan on (stock) (#blks=25001, #recs=100000)

Actual #recs: 3
```

- A query accessing multiple tables with `WHERE`

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

Actual #recs: 10
```

- A query with `ORDER BY`

```
SQL> EXPLAIN SELECT h_date, h_amount, h_data, h_c_id FROM history WHERE h_c_id < 100 ORDER BY h_date

query-plan
-----------------------------------------------------------------
->SortPlan (#blks=19, #recs=741)
        ->ProjectPlan  (#blks=859, #recs=741)
                ->SelectPlan pred:(h_c_id<100.0) (#blks=859, #recs=741)
                        ->TablePlan on (history) (#blks=859, #recs=30000)

Actual #recs: 990
```

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

```
SQL> EXPLAIN SELECT h_date, SUM(h_amount) FROM history WHERE h_c_id < 40 GROUP BY h_date

query-plan
-----------------------------------------------------------------
->ProjectPlan  (#blks=7, #recs=228)
        ->GroupByPlan: (#blks=7, #recs=228)
                ->SortPlan (#blks=7, #recs=235)
                        ->SelectPlan pred:(h_c_id<40.0) (#blks=859, #recs=235)
```

```
                                ->TablePlan on (history) (#blks=859, #recs=30000)

Actual #recs: 18
```