Assignment3 report

Group Members

1. 107062321 王劭元
2. 107062214 陳伯瑾
3. 107062228 陳劭愷

Implementation

Planner

Planner

帶有 `EXPlAIN` 的 SQL 指令會先到 `Plan::createQueryPlan` 這個函數，經過 Parser 的處理後（見下方 Parse/Parser），得到處理後的 `QueryData`，接著進到 `BasicQueryPlanner::createPlan`。

BasicQueryPlanner

`BasicQueryPlanner` 中，透過 `QueryData` 來建立 plan tree，可以看到由下而上，依序建立：

1. `TablePlan` → 負責讀取一個 Table 的資料。
2. `ProductPlan` → 當有多個 tables 被選擇時，對兩兩的 tables 進行 full join。
3. `SelectPlan` → 負責 SQL 指令的 `WHERE` 語句部分的篩選。
4. `GroupByPlan` → 負責 SQL 指令的 `GROUP BY`，以及 aggregation functions 的處理，例如 `COUNT`、`AVG`、`SUM` 等等。
5. `ProjectPlan` → 負責篩選 fields 的工作，也就是 SQL 指令的 `SELECT` 語句。
6. `SortPlan` → 負責 SQL 指令的 `ORDER BY` 指令，讓在這底下的 Plan 可以在預先排序好的 table 上面搜索下一筆資料。
7. `ExplainPlan` → 這次新加的 Plan，當 SQL 指令中有 `EXPLAIN` 語句，則建立 plan tree 的結構成一個 field `query-plan`。

```
// Step 7: Add a Explain plan if need
if(data.getIsExplain())
  p = new ExplainPlan(p);
```

Parse

Lexer

在 `Lexer::initKeywords` 中新增一個 keyword `"explain"`。

Parser

```
public QueryData queryCommand() {
  boolean isExplain = false;
  if (lex.matchKeyword("explain")) {
    lex.eatKeyword("explain");
    isExplain = true;
  }
  // ...
  return new QueryData(projs.asStringSet(), tables, pred,
      groupFields, projs.aggregationFns(), sortFields, sortDirs , isExplain);
}
```

上面提到，在 `Plan::createQueryPlan` 中，會使用 `Parser::queryCommand` 來將資料轉換為 `QueryData` 以方便分析使用，原本的 `Parser::queryCommand` 只處理了一般查詢的 SQL，這裡加上對 `EXPLAIN` 的處理，並且將其傳入 `QueryData` 中。

QueryData

在 `Parser::queryCommand` 中傳入的 `isExplain` 會變成 `QueryData` 的 private variable，因此寫一個 getter function `QueryData::getIsExplain` 讓其他 class 存許是否 SQL 中有 `EXPLAIN` 語句。

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

Algebra

ExplainPlan

`blocksAccessed()` `histogram()` `recordsOutput()` 由於Explain跟Project的性質有點像，都是在最上層把下面實作好的plan包起來的作用，因此在這三個function都直接拿下一層的plan處理好的資訊就可以了。

不一樣的有 建構子和 `open()` 由於在這次作業都已經限制好我們的schema只有一個field 叫query-plan，因此我們在這邊在建構子的部份就不用傳schema進來，並且在 `open()` 的地方幫schema加一個query-plan的field，並且return ExplainScan。

```
@Override
public Scan open() {
  Scan s = p.open();
  schema.addField("query-plan", VARCHAR(500));
  return new ExplainScan(s, schema.fields(), getExplain(0));
}
```

ExplainScan

```java
public ExplainScan(Scan s, Collection<String> fieldList, String explain) {
  this.s = s;
  this.fieldList = fieldList;
  this.explain = explain;
  this.first = true;
}
```

```java
public boolean next() {
  if (first) {
    first = false;
    return true;
  }
  return false;
}
```

```java
public Constant getVal(String fldName) {
  if (hasField(fldName))
    return Constant.newInstance(VARCHAR, explain.getBytes());
  else
    throw new RuntimeException("field " + fldName + " not found.");
}
```

我們的ExplainScan主要是參考ProjectScan來實作的。

關於`beforeFirst()`, `close()`, `hasField()`這三個function的功能，和ProjectScan的功能是相同的。

`beforeFirst()`會層層往下呼叫下面的Scan到最底層的TableScan，在TableScan中的`beforeFirst()`會指到ResultSet的第一個Record的前面，而`close()`的功能是要將scan關掉，`hasField()`的功能則是要檢查傳入的field是否存在schema中。

不同的部分在於：

建構子，`next()`和`getVal()`這三個function，以及我們多設了兩個變數`explain`(string)和`first`(boolean)，並且在建構子中對`explain`和`first`做設定。

- `explain`

  從ExplainPlan中，呼叫`open()`，並在`open()`中呼叫new ExplainScan時，從傳入的參數就可以知道`explain`這個string其實就是紀錄了執行`EXPLAIN`後的得到的結果，而在client端呼叫RemoteResultSet.getString("query-plan")時，便會依序呼叫到ExplainScan中的`getVal()`，並且將`explain`(執行`EXPLAIN`後的結果)包成一個`Constant`回傳回去。

- `first`

  在這次的作業中，會將`EXPLAIN`得到的結果全部存在第一筆record中的query-plan這個field，且ResultSet中只會有那一筆record。而`first`會在`next`這個function中被使用到，目的即在於確認是否已經讀過那唯一一筆record。

Plan

在 `Plan` 這個 interface 中新增一個 `getExplain` 函數，每一個 implement `Plan` 的 class 都必須有 `getExplain` 來輸出自己那層的 explain 資訊，因為 explain 輸出的資訊必須是有階層的，因此 `getExplain` 有一個參數 `depth` 代表現在在 plan tree 的第幾層。

- `TablePlan`

  `TablePlan` 已經是最下層的 plan 了，因此直接返回此層的 `EXPLAIN`。

```
public String getExplain(int depth) {
  String explain = "";
  for (int i = 0; i < depth; i++)
    explain += "\t";
  explain += String.format("->TablePlan on (%s) (#blks=%d, #recs=%d)\n",
    ti.tableName(), blocksAccessed(), recordsOutput());
  return explain;
}
```

- `ProductPlan`

  每一個 `ProductPlan` 都負責 Join 兩個 tables，所以返回自己的 `EXPLAIN` 以及下面兩個 table plans 的 `EXPLAIN`。

```
public String getExplain(int depth) {
  String explain = "";
  for (int i = 0; i < depth; i++)
    explain += "\t";
  explain += String.format("->ProductPlan (#blks=%d, #recs=%d)\n", blocksAccessed(), recordsOutput());
  return explain + p1.getExplain(depth + 1) + p2.getExplain(depth + 1);
}
```

- `SelectPlan`

```
public String getExplain(int depth) {
  String explain = "";
  for (int i = 0; i < depth; i++)
    explain += "\t";
  explain += String.format("->SelectPlan pred:(%s) (#blks=%d, #recs=%d)\n", pred.toString(), blocksAccessed(), recordsOutput());
  return explain + p.getExplain(depth + 1);
}
```

- `GroupByPlan`

```
public String getExplain(int depth) {
  String explain = "";
  for (int i = 0; i < depth; i++)
    explain += "\t";
  explain += String.format("->GroupByPlan: (#blks=%d, #recs=%d)\n", blocksAccessed(), recordsOutput());
  return explain + sp.getExplain(depth + 1);
}
```

- `ProjectPlan`

```
public String getExplain(int depth) {
  String explain = "";
  for (int i = 0; i < depth; i++)
    explain += "\t";
  explain += String.format("->ProjectPlan (#blks=%d, #recs=%d)\n", blocksAccessed(), recordsOutput());
  return explain + p.getExplain(depth + 1);
}
```

- `SortPlan`

```java
public String getExplain(int depth) {
  String explain = "";
  for (int i = 0; i < depth; i++)
    explain += "\t";
  explain += String.format("->SortPlan (#blks=%d, #recs=%d)\n", blocksAccessed(), recordsOutput());
  return explain + p.getExplain(depth + 1);
}
```

- `ExplainPlan`

  最後在 `ExplainPlan` 中輸出 actual records。

```java
public String getExplain(int depth) {
  String str = "\n" + p.getExplain(depth) + String.format("Actual: #recs = %d", p.recordsOutput());
  return str;
}
```

Worth Mentioning

1. 這裡我們去嘗試join了三個table後，再去select出c_id=4的record，雖然我們最後成功跑出來了，但可以看到中間在做完第二次product後，#recs來到了 9×10^{13} 個。但只要我們像老師上課提的，把select往下先做後，再去join的話，就可以不會join後，一次要access這麼多個record。因此這應該是之後會去慢慢修改的地方！

```
SQL> EXPLAIN SELECT c_id FROM customer, history, item WHERE c_id=4

query-plan
-----------------------------------------------------------------------------------
->ProjectPlan (#blks=45003085906251, #recs=101166097981)
        ->SelectPlan pred:(c_id=4.0) (#blks=45003085906251, #recs=101166097981)
                ->ProductPlan (#blks=45003085906251, #recs=90000000000000)
                        ->ProductPlan (#blks=85906251, #recs=3000000000)
                                ->TablePlan on (item) (#blks=6251, #recs=100000)
                                ->TablePlan on (history) (#blks=859, #recs=30000)
                        ->TablePlan on (customer) (#blks=15001, #recs=30000)
Actual: #recs = 101166097981
```

Demo

- A query accessing single table with WHERE

```
SQL> EXPLAIN SELECT c_first FROM customer WHERE c_id > 3;

query-plan
-----------------------------------------------------------------------
->ProjectPlan (#blks=15001, #recs=29986)
        ->SelectPlan pred:(c_id>3.0) (#blks=15001, #recs=29986)
                ->TablePlan on (customer) (#blks=15001, #recs=30000)
Actual: #recs = 29986
```

- A query accessing multiple tables with WHERE

```
SQL> EXPLAIN SELECT h_data FROM history, customer WHERE c_id=h_c_id;

query-plan
-----------------------------------------------------------------------------
->ProjectPlan (#blks=450030859, #recs=1010231)
        ->SelectPlan pred:(c_id=h_c_id) (#blks=450030859, #recs=1010231)
                ->ProductPlan (#blks=450030859, #recs=900000000)
                        ->TablePlan on (history) (#blks=859, #recs=30000)
                        ->TablePlan on (customer) (#blks=15001, #recs=30000)
Actual: #recs = 1010231
```

- A query with ORDER BY

```
SQL> EXPLAIN SELECT i_price FROM item WHERE i_price>50 ORDER BY i_price;

query-plan
-------------------------------------------------------------------------
->SortPlan (#blks=189, #recs=48039)
        ->ProjectPlan (#blks=6251, #recs=48039)
                ->SelectPlan pred:(i_price>50.0) (#blks=6251, #recs=48039)
                        ->TablePlan on (item) (#blks=6251, #recs=100000)
Actual: #recs = 48039
```

- A query with GROUP BY and at least one aggregation function

```
SQL> EXPLAIN SELECT d_id, COUNT(h_d_id) FROM district, history WHERE d_id=h_d_id GROUP BY d_id;

query-plan
-----------------------------------------------------------------------------------------------
->ProjectPlan (#blks=1409, #recs=10)
        ->GroupByPlan: (#blks=1409, #recs=10)
                ->SortPlan (#blks=1409, #recs=12679)
                        ->SelectPlan pred:(d_id=h_d_id) (#blks=8592, #recs=12679)
                                ->ProductPlan (#blks=8592, #recs=300000)
                                        ->TablePlan on (district) (#blks=2, #recs=10)
                                        ->TablePlan on (history) (#blks=859, #recs=30000)
Actual: #recs = 10
```