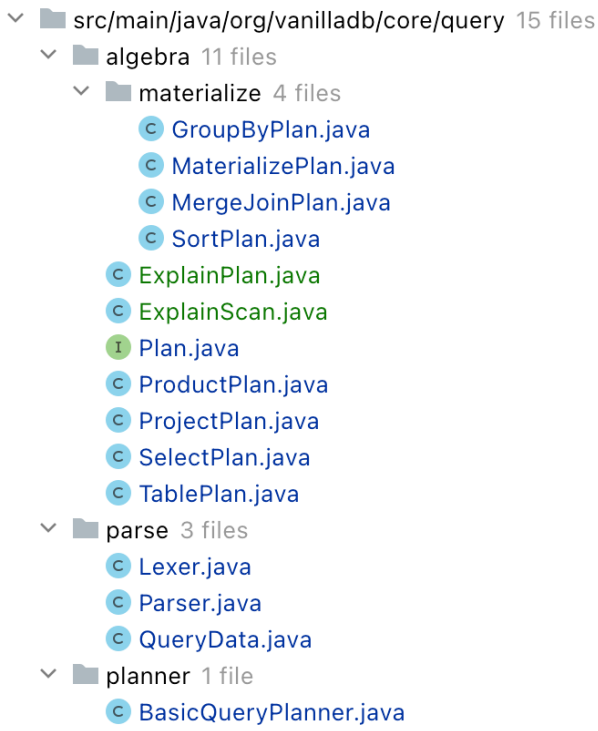
3_assignment3_report1

Group Members Student IDs:

- 108000208 王彥婷
- 108000115 李思澔
- 108062228 高以遐

Implementation

Modified/Added Classes



Parse

1. 在 `Lexer.java` 中新增 explain keyword，之後便可以利用 `Lexer` 來判斷 `EXPLAIN` 為 keyword。

```
// org.vanilladb.core.query.parse.Lexer
...
private void initKeywords() {
  keywords = Arrays.asList(..., "explain");
}
```

2. 在 `QueryData.java` 的 constructor 中新增 `explain` 的 field，目的是讓 `BasicQueryPlanner` 能夠判斷需不需要加上 `ExplainPlan`。同時修改 `toString()` function 使得在呼叫該 function 的時候也會印出 explain。

```
// org.vanilladb.core.query.parse.QueryData
...
private boolean explain;
...
public QueryData(..., boolean explain) {
  ...
  this.explain = explain;
}
...
public boolean getExplain() {
  return explain;
}
```

3. 在 `Parser` 的 `queryCommand()` function 中，透過 `lex.matchKeyword` 來檢查這個 query 是否需要印出 explain，並以 boolean 型別的參數紀錄，並將該參數傳入並生成 QueryData 物件。

```
// org.vanilladb.core.query.parse.Parser
...
public QueryData queryCommand() {
  boolean explain = false;
  if (lex.matchKeyword("explain")) {
    explain = true;
    lex.eatKeyword("explain");
  }
  ...
  return new QueryData(..., explain);
}
```

Planner

1. `BasicQueryPlanner` 根據 `QueryData` 提供的 `getExplain()` 決定要不要在 explain tree 的最上層面加上 `ExplainPlan`。

```
// org.vanilladb.core.query.planner.BasicQueryPlanner
...
public Plan createPlan(QueryData data, Transaction tx) {
  ...
  if (data.getExplain()) {
    p = new ExplainPlan(p);
  }
  return p;
}
```

Algebra

1. 在 `Plan` 中新增一個 method 用來產生 explain，所有 implement Plan 的 class 都需要實做 `generateExplanation()` 這個 method，並且傳入參數 `level` 來處理縮排的問題。

```
// org.vanilladb.core.query.algebra.Plan
...
String generateExplanation(int level);
```

2. 所有的 `Plan` 都需要 implements 上面的 `generateExplanation(int level)` method，作法大同小異，這邊以 `ProductPlan` 與 `SelectPlan` 為例。我們透過 level 參數來處理縮排的問題，會根據目前的 level 計算要在字串前面加入多少空格，並在繼續呼叫 subplan 的時候將 level 加一。

   - `ProductPlan`: 通過原本就有的 `blocksAccessed()` 以及 `recordsOutput()`，取得 explain 中需要的資料，再 recursively 將 sub-plan 的 explanation 也產生出來。

     ```
     // org.vanilladb.core.query.algebra.ProductPlan
     ...
     public String generateExplanation(int level) {
       String explanation = String.format("->ProductPlan (#blks=%d, #recs=%d)%n", blocksAcces
     sed(), recordsOutput());
       explanation = new String(new char[level]).replace("\0", "  ") + explanation;
       explanation += p1.generateExplanation(level + 1);
       explanation += p2.generateExplanation(level + 1);
       return explanation;
     }
     ```

   - `SelectPlan`: 除了 `blocksAccessed()` 以及 `recordsOutput()` 還會透過 `pred.toString()` 來取得 query 中的 predicate。

     ```
     // org.vanilladb.core.query.algebra.SelectPlan
     ...
     public String generateExplanation(int level) {
       String explanation = String.format("->SelectPlan pred:(%s) (#blks=%d, #recs=%d)%n",
           pred.toString(), blocksAccessed(), recordsOutput());
       explanation = new String(new char[level]).replace("\0", "  ") + explanation;
       explanation += p.generateExplanation(level + 1);
       return explanation;
     }
     ```

   - 透過類似方法修改的檔案有 `GroupByPlan`、`MaterializePlan`、`MergeJoinPlan`、`SortPlan`、`ProjectPlan`、`TablePlan`（基本上就是所有 implement `Plan` 這個 class 的檔案）

3. `ExplainPlan` 主要用途是用 `open()` 產生以 `ExplainScan` 為根的 scan tree。首先在這個 Plan 下產生一個 schema，其中 Field 為 query-plan 且紀錄的型別為 `VARCHAR(500)` 的資料。另外 `blocksAccessed()`、`histogram()`、`generateExplanation(int level)` 等 function 都直接回傳 subplan 的結果，而 `recordsOutput()` 則回傳 1，因為該 table 中只有一個 record。

   ```
   // org.vanilladb.core.query.algebra.ExplainPlan
   ...
   public ExplainPlan(Plan p) {
     ...
     schema.addField("query-plan", Type.VARCHAR(500));
   }
   ```

```
...
public Scan open() {
  return new ExplainScan(p, schema);
}
```

4. 在 `ExplainScan` 的 constructor 中，計算真正的 records 的數量，生成 explanation 並且存起來，在 `getVal(string fieldName)` 中使用。另外新增一個 count 參數紀錄最後得到多少 records，記算的方法為不斷呼叫 s.next() 來計算最後得到的 record 總數。

```
// org.vanilladb.core.query.algebra.ExplainScan
...
public ExplainScan(Plan p, Schema schema) {
  ...
  this.explanation = String.format("%n") + p.generateExplanation(0) + String.format("%nActua
l #recs: %d", count);
}
...
public Constant getVal(String fldName) {
  if (hasField(fldName)) {
    return new VarcharConstant(explanation);
  } else {
    throw new RuntimeException("field " + fldName + " not found.");
  }
}
```

另外在實作 `next()` method 的時候，因為只有一個 record，因此只有第一次呼叫 `next()` 的時候會回傳 true，實作方法為透過一個 `count` 參數，於呼叫 `beforefirst()` 將該參數設為 0，並在呼叫 `next()` 的時候將 count 加一，若呼叫 `next()` 時 count > 1，就會回傳 false。

Experiment

Queries accessing single table with `WHERE`

Query1:

```
EXPLAIN SELECT COUNT(c_payment_cnt) FROM customer WHERE c_payment_cnt = 1
```

- result

```
query-plan
------------------------------------------------------------------------
->ProjectPlan (#blks=15001, #recs=1)
  ->GroupByPlan (#blks=15001, #recs=1)
    ->SelectPlan pred:(c_payment_cnt=1.0) (#blks=15001, #recs=6936)
      ->TablePlan on (customer) (#blks=15001, #recs=30000)

Actual #recs: 1
```

Query2:

```
EXPLAIN SELECT c_payment_cnt FROM customer WHERE c_payment_cnt = 1
```

- result

```
query-plan
-----------------------------------------------------------------
->ProjectPlan (#blks=15001, #recs=6936)
  ->SelectPlan pred:(c_payment_cnt=1.0) (#blks=15001, #recs=6936)
    ->TablePlan on (customer) (#blks=15001, #recs=30000)

Actual #recs: 30000
```

A query accessing multiple tables with `WHERE`

Query1

```
EXPLAIN SELECT COUNT(c_id) FROM customer, district, warehouse WHERE c_d_id = d_id and c_w_id = w_id
```

- result

```
query-plan
-----------------------------------------------------------------
->ProjectPlan (#blks=150032, #recs=1)
  ->GroupByPlan (#blks=150032, #recs=1)
    ->SelectPlan pred:(c_d_id=d_id and c_w_id=w_id) (#blks=150032, #recs=2914)
      ->ProductPlan (#blks=150032, #recs=300000)
        ->ProductPlan (#blks=22, #recs=10)
          ->TablePlan on (district) (#blks=2, #recs=10)
          ->TablePlan on (warehouse) (#blks=2, #recs=1)
        ->TablePlan on (customer) (#blks=15001, #recs=30000)

Actual #recs: 1
```

Queries with `ORDER BY`

Query1

```
EXPLAIN SELECT d_name FROM district ORDER BY d_name
```

- result

```
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
```

Query2

```
EXPLAIN SELECT w_tax FROM warehouse ORDER BY w_tax
```

- result

```
query-plan
-------------------------------------------------------------------
->SortPlan (#blks=1, #recs=1)
  ->ProjectPlan (#blks=2, #recs=1)
    ->SelectPlan pred:() (#blks=2, #recs=1)
      ->TablePlan on (warehouse) (#blks=2, #recs=1)

Actual #recs: 1
```

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

Query1

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

- result

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

Actual #recs: 1
```

Anything worth mentioning

- `algebra/materialize` 中所有 Plan 裡面的 `toString()` method 已經實作完成，似乎是解答的程式碼沒有刪除乾淨。