Assignment 3 — Phase 1

> Team 11
> 107062303 劉緒紳、107062120 姚潔恩、107062372 張凱博

Implementation of Explain operation

package `parse`

class `Lexer`

首先我們在 keywords list 中加入 `Explain` 這個 keyword。

class `QueryData`

接著我們在 `QueryData` 這個 class 中增加 `this.explaining` 這個 attribute，type 為 `boolean`，表示這個 query 是不是 explain operation。

class `Parser`

在 parse query command 的時候增加對 explain 的處理。若有 match 到 explain 這個 keyword，傳給 `QueryData` constructor 的 `explaining` 就會設定為 true，表示這個 query 為 explain operation。

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

    // select, from, where, ...

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

```
        groupFields, projs.aggregationFns(), sortFields, sortDirs, explaining);
  }
```

package `planner`

class `BasicQueryPlanner`

Query 被 Parse 完後得到的 `QueryData` 會被送給 `BasicQueryPlanner` 的 function `createPlan` 以產生 query plan。其中 `createPlan` 原本有 6 個 step，而我們需要在最後檢查這個 query 是不是 explain operation，如果是的話就再 wrap 一層 `ExplainPlan`。

```
@Override
public Plan createPlan(QueryData data, Transaction tx) {
  // Step 1: Create a plan for each mentioned table or view
  // ...
  // Step 2: Create the product of all table plans
  // ...
  // Step 3: Add a selection plan for the predicate
  // ...
  // Step 4: Add a group-by plan if specified
  // ...
  // Step 5: Project onto the specified fields
  // ...
  // Step 6: Add a sort plan if specified
  // ...

  if (data.isExplaining()) {
    // wrap p with ExplainPlan
    p = new ExplainPlan(p);
  }

  return p;
}
```

package `algebra`

class `ExplainPlan`

就像前面 `createPlan` 提到的，`ExplainPlan` 會將目標 Plan 包裝起來，並提供 explain 的功能。

根據作業的說明，我們要將 Explain 的結果放在 `query-plan` 這個 column 中，而其型別為 `varchar(500)`，因此我們直接在 `ExplainPlan` 的 constructor 中設定他的 schema：

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

接著是最重要的部分：建構對這道 SQL Query 進行 Explain 的結果。

我們的做法是，在 `Plan` 這個 interface 中加入一個方法 `getExplainString(level)` ，回傳這個 Plan 的資訊，而 level 是決定這個 Plan 位於第幾層，前方需要加多少空白。

例如，位於最內層的 `TablePlan` 就可以這麼實作：

```
@Override
public String getExplainString(int level) {
    // repeat "\t" level times
    String tabs = new String(new char[level]).replace("\0", "\t");
    return tabs + String.format("->TablePlan on (%s) (#blks=%d, #recs=%d)\n",
           ti.tableName(), blocksAccessed(), recordsOutput());
}
```

這會回傳類似這樣的字串：

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

而其他不會在最內層的 Plan 則可以透過類似遞迴的方式呼叫下一層 Plan 的 `getExplainString` 函式，例如 `ProductPlan` 會呼叫兩個要被 product 的 Plan 的 `getExplainString` ：

```
@Override
public String getExplainString(int level) {
    // repeat "\t" level times
    String tabs = new String(new char[level]).replace("\0", "\t");
    return tabs
           + String.format("->ProductPlan (#blks=%d, #recs=%d)\n",
                           blocksAccessed(), recordsOutput())
           + p1.getExplainString(level + 1)
           + p2.getExplainString(level + 1);
  }
```

其他每個 Plan 的實作方式皆相似，這裡就不再贅述了。

而 `ExplainPlan` 除了建立每個 Plan 的資訊外，還要計算該 Query 真正會回傳的 record 數量，這項功能我們直接透過重複呼叫 `scan.next()` 來得到總共會有多少筆 record：

```
private int getActualRecs() {
    Scan s = p.open();
    s.beforeFirst();
    int recs = 0;
    while (s.next()) {
        recs += 1;
    }
    return recs;
}
```

最後我們在 `open` 方法中將建構好的 Explain 字串傳給 `ExplainScan`：

```
@Override
public String getExplainString(int level) {
    return "\n" + p.getExplainString(0) +
           String.format("\nActual #recs: %d", getActualRecs());
}

@Override
public Scan open() {
    String explain_string = getExplainString(0);
    ExplainScan s = new ExplainScan(schema.fields(), explain_string);
    return s;
}
```

class `ExplainScan`

完成 `ExplainPlan` 後，我們就可以實作 `ExplainScan` 來將 Explain 的結果以 Iterator 的方式回傳給外界。

具體來說，因為 Explain 的結果只有一個 row，所以我們是用一個 counter 來模擬 Iterator 的運作：當 `beforeFirst` 被呼叫時，將 counter 設為 0，而當 next 被呼叫時則將 counter + 1 代表 Scan 裡的資料被取出了。

```
public ExplainScan(Collection<String> fieldList, String es) {
    this.fieldList = fieldList;
    explain_string = es;
    counter = 0;
}

@Override
```

```
public void beforeFirst() {
    counter = 0;
}

@Override
public boolean next() {
    if (counter > 0) {
        return false;
    }
    counter++;
    return true;
}
```

最後實作 `getVal` 來將我們建構出的 Explain 結果回傳出去：

```
public Constant getVal(String fldName) {
    if (hasField(fldName))
        return new VarcharConstant(explain_string);
    else
        throw new RuntimeException("field " + fldName + " not found.");
}
```

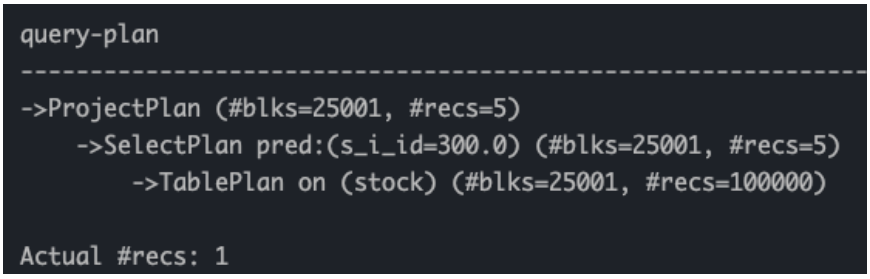
如此一來就完成了 Explain 功能的實作。

The `EXPLAIN` result for the following queries

- A query accessing single table with `WHERE`
  - Command:

    ```
    explain select s_dist_01, s_dist_02, s_dist_03, s_dist_04, s_dist_05
    from stock
    where s_i_id = 300;
    ```

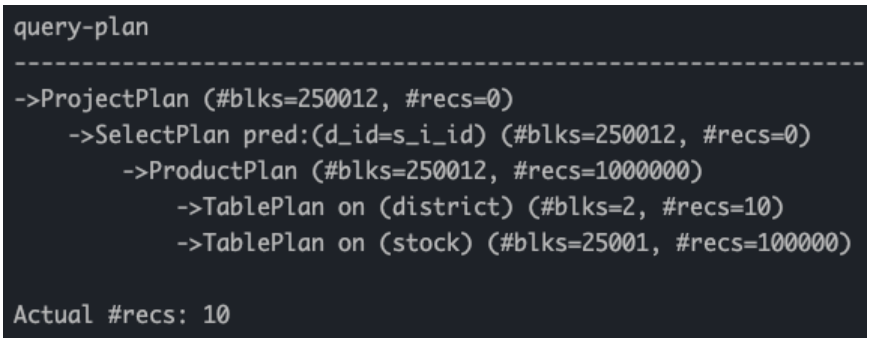
  - Result Screenshot:



- A query accessing multiple tables with `WHERE`
  - Command:

```
explain select d_id, d_name, s_i_id
from district, stock
where d_id=s_i_id;
```

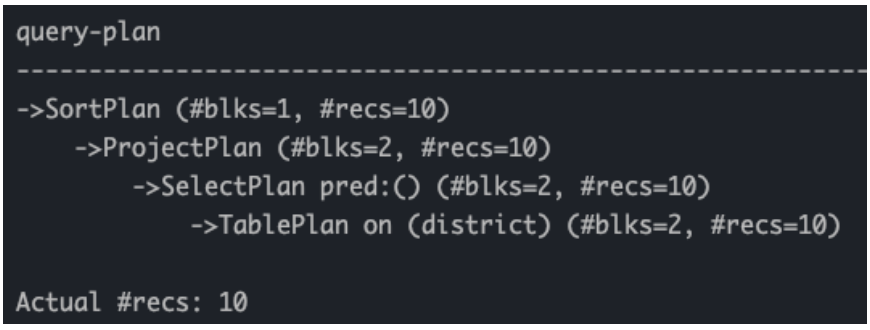
  - Result Screenshot:



- A query with `ORDER BY`
  - Command:

```
explain select d_id
from district
order by d_id desc;
```

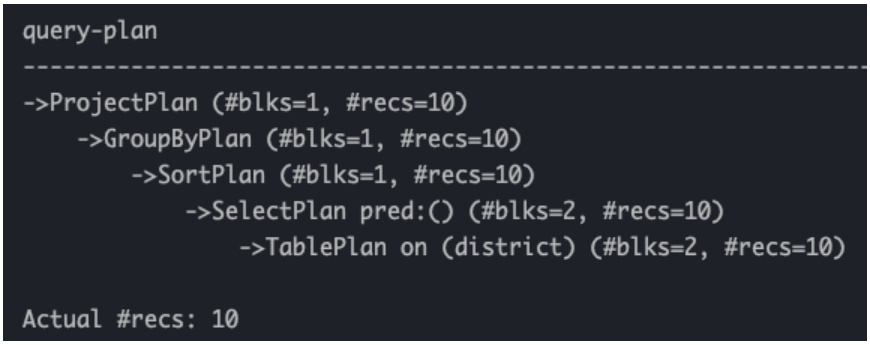
  - Result Screenshot:



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

```
explain select count(d_id), d_name
from district
group by d_name;
```

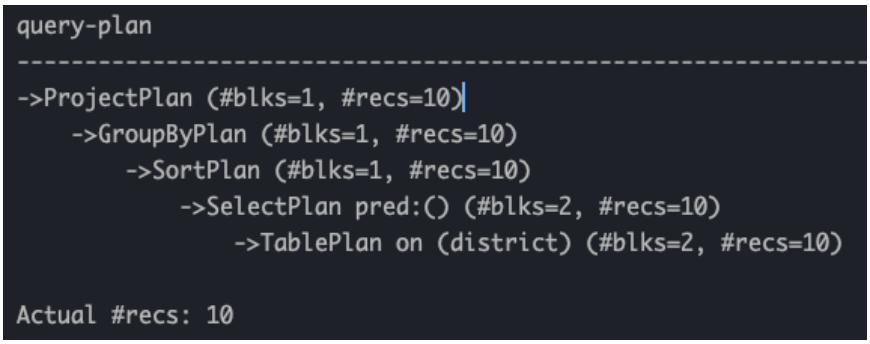
  - Result Screenshot:



- First additional queries experiments (`GROUP BY` and `SUM`)
  - Command:

```
explain select d_name, sum(d_tax)
from district
group by d_name;
```

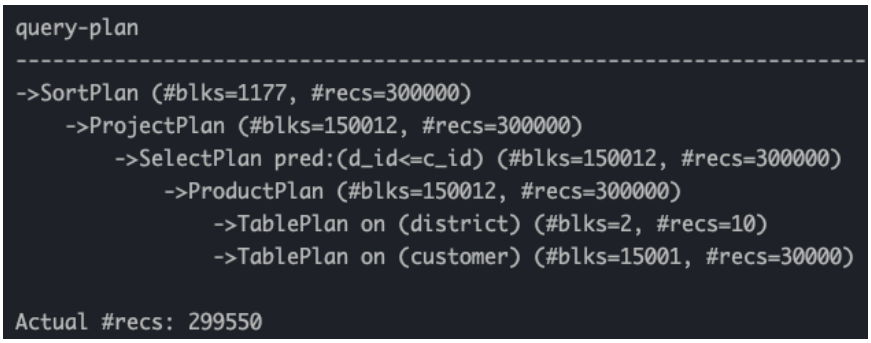
  - Result Screenshot:



- Second additional queries experiments (`WHERE` and `ORDER BY`)
  - Command:

```
explain select d_id, c_id
from district, customer
where d_id <= c_id
order by d_id;
```

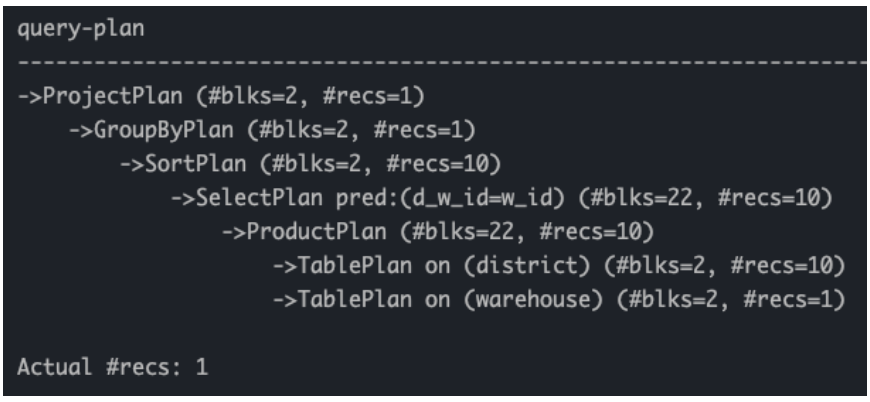
  - Result Screenshot:



- Third additional queries experiments (`COUNT`, `WHERE`, and `GROUP BY`)
  - Command:

```
explain select count(d_id)
from district, warehouse
where d_w_id = w_id
group by w_id;
```

  - Result Screenshot:



- Forth additional queries experiments (`COUNT`, `GROUP BY`, and `ORDER BY`)
  - Command:

```
explain select count(d_id), d_name
from district
group by d_name
order by count(d_id);
```

  - Result Screenshot: