Assignment 3 Phase 1 Report

110062229 翁語辰
110081014 程詩柔
110062171 陳彥成

1. Implementation

- org.vanilladb.core.query.algebra.Plan.java

  在Plan中新增toString(int level)的method。然後在他的實作類都重寫細節的實現，讓每個plan都可以印出關於自己的內容

```
String toString(int level);
```

- org.vanilladb.core.query.parse.Lexer.java

  在最下面initKeywords()中加入”explain”這個keyword

- org.vanilladb.core.query.parse.Parser.java

  在Parser.java的queryCommand()中，我們新增先檢查explain的邏輯，設置explain=true供後續使用，再吃掉explain

```
public QueryData queryCommand() {
    boolean explain = false;
    if (lex.matchKeyword(keyword:"explain")) {
        explain = true;
        lex.eatKeyword(keyword:"explain");
    }
    lex.eatKeyword(keyword:"select");
    ProjectList projs = projectList();
```

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

  如果在parsing的時候有吃到”explain”的話，在createPlan的最後要結束前就用explainPlan包在最外層

```
// Step 7: Add a explain plan if specified
if (data.explain() == true)
    p = new ExplainPlan(p);
return p;
```

- org.vanilladb.core.query.algebra.ExplainPlan.java

  ExplainPlan的實作與其他plan大致相同，主要就是open的時候要回傳的是ExplainScan，被call toString的時候由於不用output出ExplainPlan相關的資訊因此只需要呼叫下一層的toString就好，schema的話只有一個叫做query-plan的field

```
public class ExplainPlan implements Plan {
    private Plan p;

    public ExplainPlan(Plan p) {
        this.p = p;
    }

    public Scan open() {
        return new ExplainScan(this.p.open(), this.schema(), toString(level:0));
    }

    public String toString(int level) {
        return p.toString(level:0);
    }

    public long blocksAccessed() {
        return this.p.blocksAccessed();
    }

    public Schema schema() {
        Schema schema = new Schema();
        schema.addField(fldName:"query-plan", Type.VARCHAR(arg:500));
        return schema;
    }

    public Histogram histogram() {
        return this.p.histogram();
    }

    public long recordsOutput() {
        return 1L;
    }
}
```

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

  ExplainScan在創建的時候由於這個String explain就是自己query-plan要存的data，所以就把result設成spec給定的樣式

```java
public ExplainScan(Scan s, Schema schema, String explain) {
    this.schema = schema;
    s.beforeFirst();

    while (s.next()) {
        ++this.numRecs;
    }

    s.close();
    this.result = "\n" + explain + "\nActual #recs: " + this.numRecs;
    this.isBeforeFirsted = true;
}
```

  ExplainScan的getVal method就是當fldName是唯一的field query-plan的時候回傳result這個varchar

```java
public Constant getVal(String fldName) {
    if (fldName.equals("query-plan")) {
        return new VarcharConstant(this.result);
    } else {
        throw new RuntimeException("field " + fldName + " not found.");
    }
}
```

- org.vanilladb.core.query.algebra.ProductPlan.java

  由於在Plan.java的時候新增的一個toString(int level)的method，因此每個基於Plan的實作都要把toString implement出來。這裡舉ProductPlan toString的實作為例，其他Plan的toString實作都一樣是照著spec的要求return對應字串。

  由於ProductPlan裡會有兩個plan，所以先根據level(第幾層就是level幾)決定要多少的縮排，接著加上ProductPlan要的資訊，最後再加上兩個Plan自己的toString並且要pass level加一進去

```java
public String toString(int level) {
    String str = "";
    for (int i = 0; i < level; i++)
        str = str + "\t";
    return str + "->ProductPlan  (#blks=" + blocksAccessed() + ", #recs=" + recordsOutput() + ")\n"
            + p1.toString(level + 1) + p2.toString(level + 1);
}
```

- org.vanilladb.core.util.ConsoleSQLInterpreter.java

  在doQuery中print records的地方，當cmd是explain開頭的話，就印出對唯一一個field query-plan的data

```java
// print records
while (rs.next()) {
    for (int i = 1; i <= numcols; i++) {
        String fldname = md.getColumnName(i);
        int fldtype = md.getColumnType(i);
        String fmt = "%" + md.getColumnDisplaySize(i);
        if (cmdf.startsWith("EXPLAIN"))
            System.out.format("%s", rs.getString(fldname));
        else {
            if (fldtype == Types.INTEGER)
                System.out.format(fmt + "d", rs.getInt(fldname));
            else if (fldtype == Types.BIGINT)
                System.out.format(fmt + "d", rs.getLong(fldname));
            else if (fldtype == Types.DOUBLE)
                System.out.format(fmt + "f", rs.getDouble(fldname));
            else
                System.out.format(fmt + "s", rs.getString(fldname));
        }
    }
    System.out.println();
}
```

2. Query Result

- A query accessing single table with `WHERE`

```
SQL> EXPLAIN SELECT COUNT(d_id) FROM district WHERE d_id>0

query-plan
-----------------------------------------------------------
->ProjectPlan  (#blks=4, #recs=1)
        ->GroupByPlan  (#blks=4, #recs=1)
                ->SelectPlan pred:(d_id>0.0) (#blks=4, #recs=30)
                        ->TablePlan on (district) (#blks=4, #recs=30)

Actual #recs: 1

SQL>
```

- A query accessing multiple tables with `WHERE`
- A query with `ORDER BY`

```
SQL> EXPLAIN SELECT COUNT(d_id) FROM district, warehouse WHERE d_w_id = w_id ORDER BY w_id DESC

query-plan
-----------------------------------------------------------
->SortPlan  (#blks=1, #recs=1)
       ->ProjectPlan  (#blks=64, #recs=1)
               ->GroupByPlan  (#blks=64, #recs=1)
                       ->SelectPlan pred:(d_w_id=w_id) (#blks=64, #recs=90)
                               ->ProductPlan  (#blks=64, #recs=90)
                                       ->TablePlan on (district) (#blks=4, #recs=30)
                                       ->TablePlan on (warehouse) (#blks=2, #recs=3)

Actual #recs: 1

SQL>
```

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

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

query-plan
-----------------------------------------------------------
->ProjectPlan  (#blks=15, #recs=1)
       ->GroupByPlan  (#blks=15, #recs=1)
               ->SortPlan  (#blks=15, #recs=90)
                       ->SelectPlan pred:(d_w_id=w_id) (#blks=64, #recs=90)
                               ->ProductPlan  (#blks=64, #recs=90)
                                       ->TablePlan on (district) (#blks=4, #recs=30)
                                       ->TablePlan on (warehouse) (#blks=2, #recs=3)

Actual #recs: 1

SQL>
```