

Assignment 3 Solution

Introduction to Database Systems

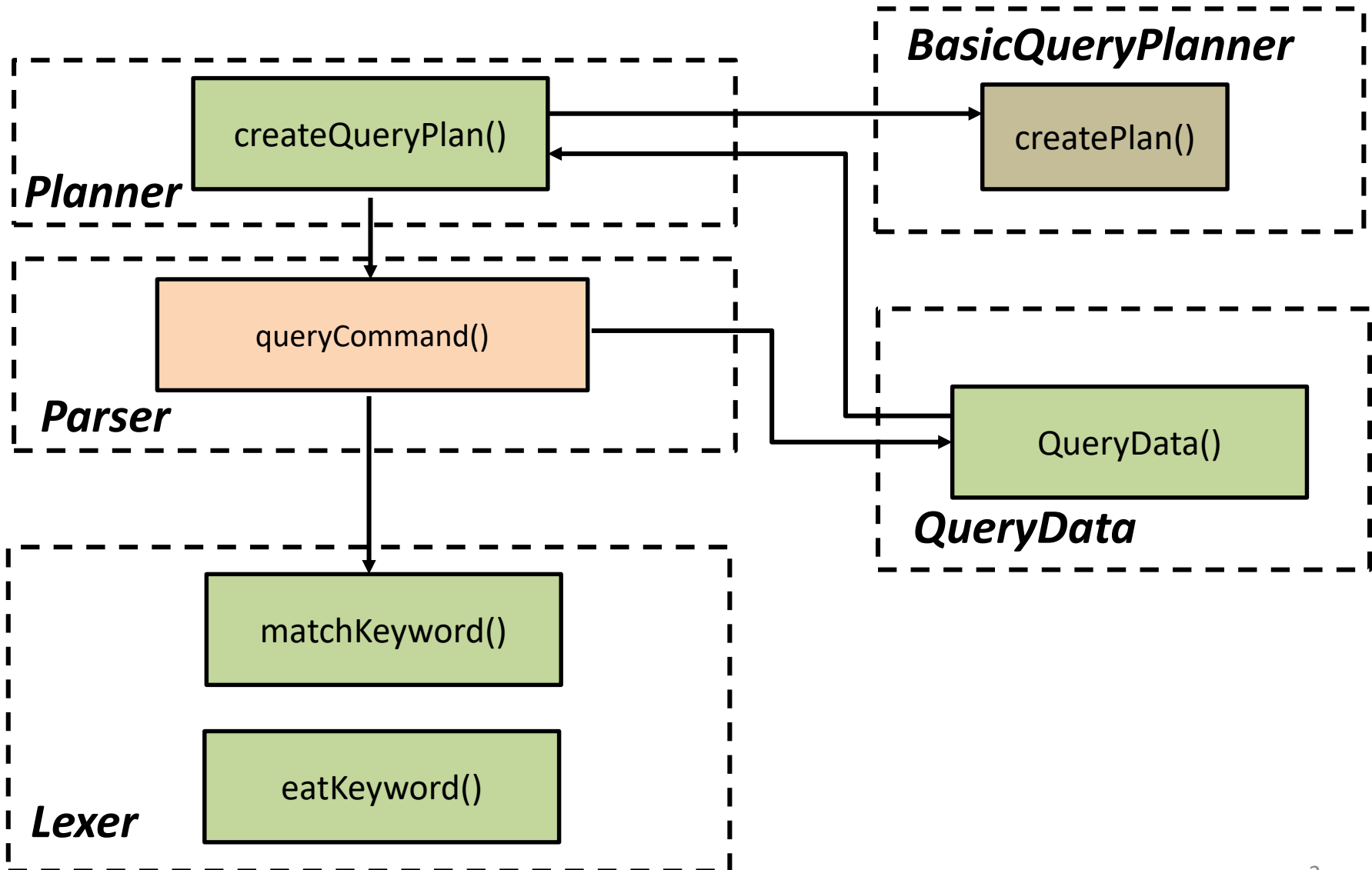
DataLab

CS, NTHU

Modified/**Added** Classes

- Parse
 - *Lexer*
 - *Parser*
 - *queryData*
- Algebra
 - *ExplaiunPlan* 、 *ExplainScan*
 - *TablePlan* 、 *ProductPlan* 、 *SelectPlan* 、 *SortPlan* 、
GroupByPlan 、 *ProjectPlan*
- Planner
 - *BasicQueryPlanner*
- An example of Experiment Results

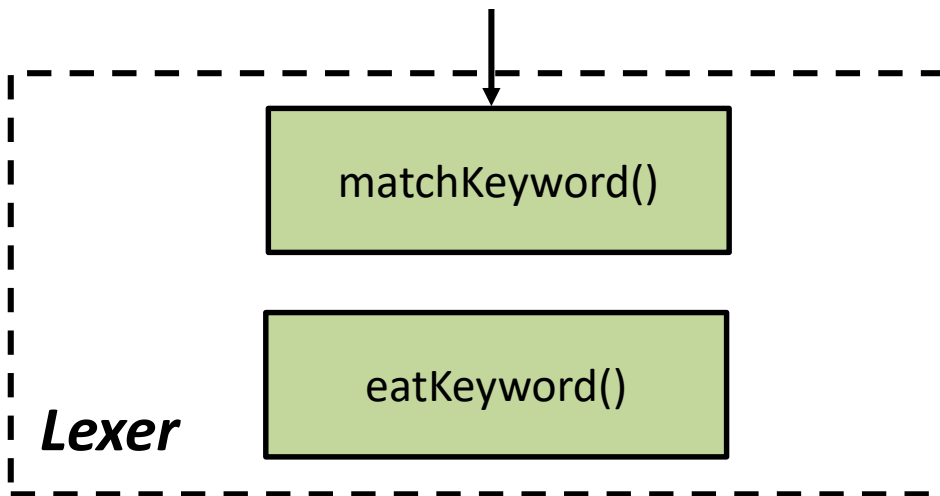
Overview



Modified/**Added** Classes

- Parse
 - *Lexer*
 - *Parser*
 - *queryData*
- Algebra
 - *ExplaiunPlan* 、 *ExplainScan*
 - *TablePlan* 、 *ProductPlan* 、 *SelectPlan* 、 *SortPlan* 、
GroupByPlan 、 *ProjectPlan*
- Planner
 - *BasicQueryPlanner*
- An example of Experiment Results

Lexer

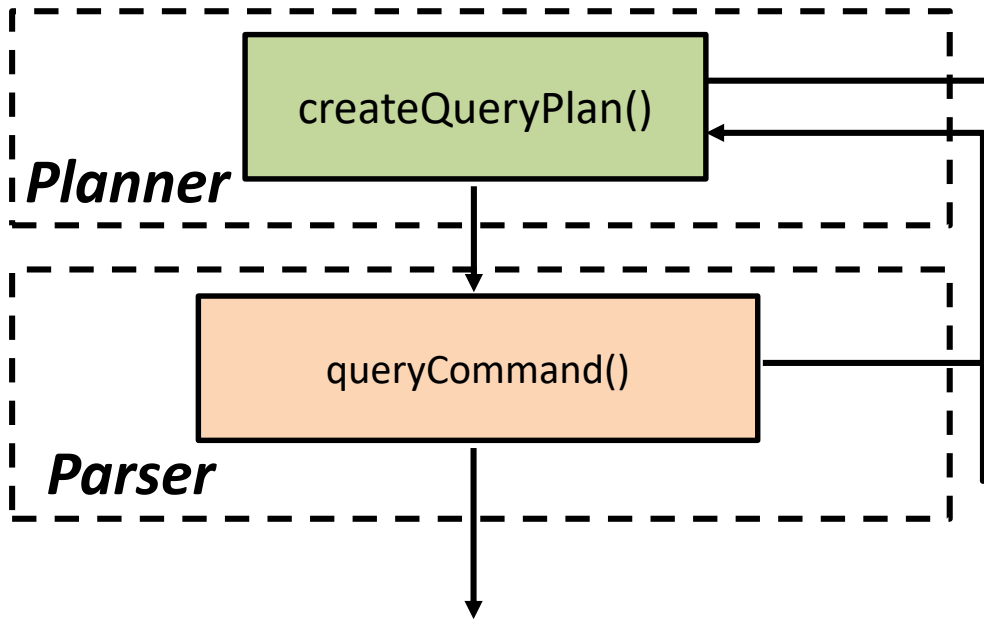


Parse

- Lexer
 - added “explain” in keywords.

```
private void initKeywords() {  
    keywords = Arrays.asList("select", "from", "where", "and", "insert",  
        "into", "values", "delete", "drop", "update", "set", "create", "table",  
        "int", "double", "varchar", "view", "as", "index", "on",  
        "long", "order", "by", "asc", "desc", "sum", "count", "avg",  
        "min", "max", "distinct", "group", "add", "sub", "mul", "div",  
        "explain", "using", "hash", "btree");  
}
```

Parser



Parse

- Parser
 - add isExplain

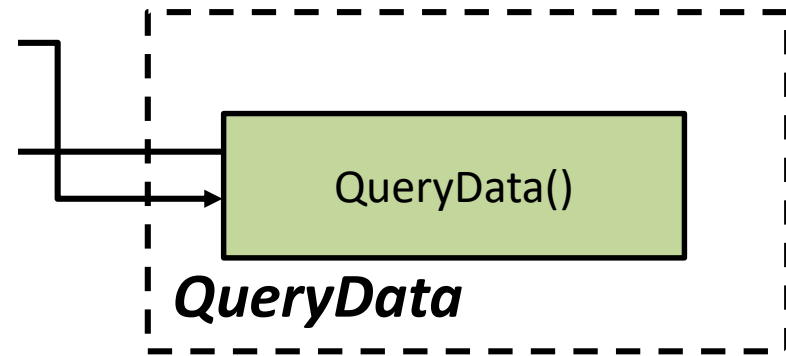
```
public QueryData queryCommand() {  
    boolean isExplain = false;  
    if (lex.matchKeyword("explain")) {  
        isExplain = true;  
        lex.eatKeyword("explain");  
    }  
    lex.eatKeyword("select");  
    ProjectList projs = projectList();  
}
```


Parse

- Parser
 - Parser returns SQL data
 - In method “queryCommand()”

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

QueryData



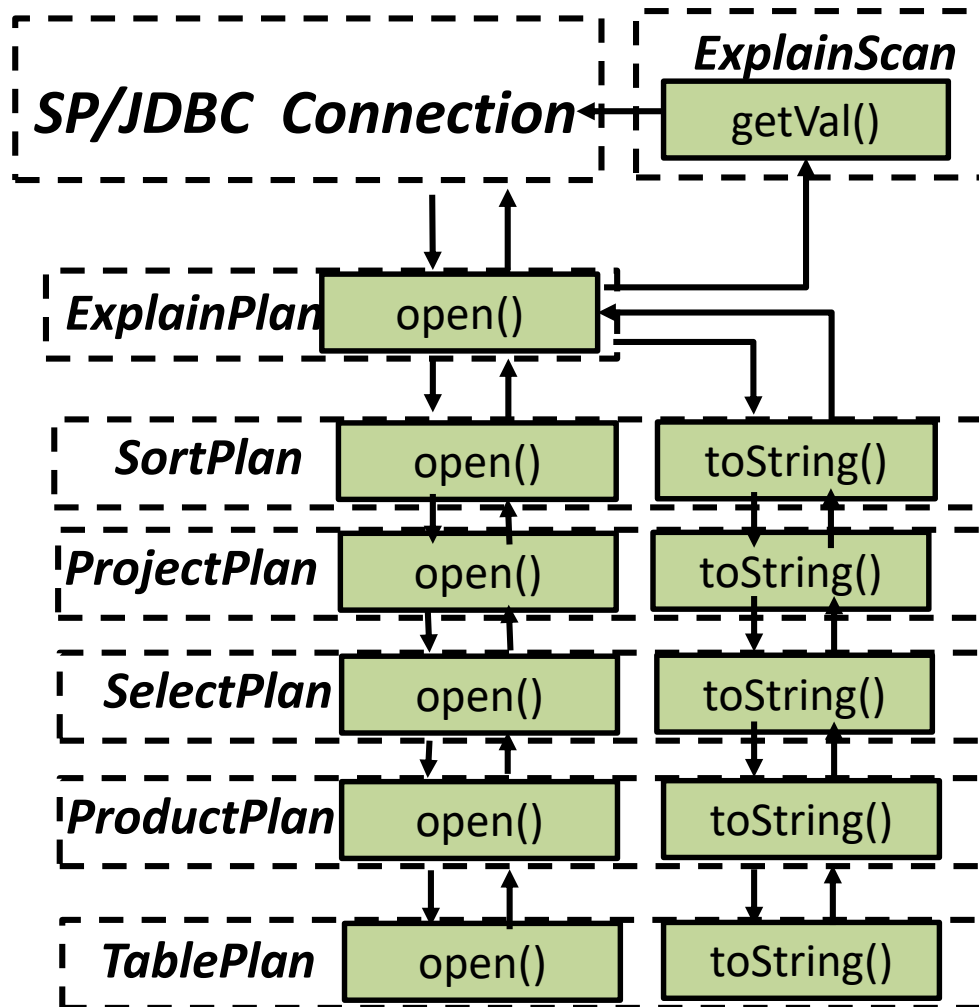
Parse

- QueryData

```
public QueryData(boolean isExplain, Set<String> projFields, Set<String> tables, Predicate pred,  
    Set<String> groupFields, Set<AggregationFn> aggFn, List<String> sortFields, List<Integer> sortDirs) {  
    this.isExplain = isExplain;  
    this.projFields = projFields;  
    this.tables = tables;  
    this.pred = pred;  
    this.groupFields = groupFields;  
    this.aggFn = aggFn;  
    this.sortFields = sortFields;  
    this.sortDirs = sortDirs;  
}
```

Modified/**Added** Classes

- Parse
 - *Lexer*
 - *Parser*
 - *queryData*
- Algebra
 - *ExplaiunPlan* 、 *ExplainScan*
 - *TablePlan* 、 *ProductPlan* 、 *SelectPlan* 、 *SortPlan* 、 *GroupByPlan* 、 *ProjectPlan*
- Planner
 - *BasicQueryPlanner*
- An example of Experiment Results



ExplainPlan

```
@Override
public Schema schema() {
    Schema schema = new Schema();
    schema.addField("query-plan", Type.VARCHAR(500));
    return schema;
}
```

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

ExplainScan

- That the result shows once

```
@Override
public void beforeFirst() {
    isBeforeFirst = true;
}

@Override
public boolean next() {
    if (isBeforeFirst) {
        isBeforeFirst = false;
        return true;
    } else
        return false;
}
```

ExplainScan

- Return the result of explain

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


ExplainScan

- Return the number of actual records

```
public ExplainScan(Scan s, Schema schema, String explain) {  
    this.result = "\n" + explain;  
    this.schema = schema;  
    s.beforeFirst();  
    while (s.next())  
        numRecs++;  
    s.close();  
    this.result = result + "\nActual #recs: " + numRecs;  
}
```

TablePlan

```
@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("->TablePlan on ").append(ti.tableName())
      .append(") (#blks=");
    sb.append(blocksAccessed()).append(", #recs=").append(recordsOutput())
      .append(")\n");
    return sb.toString();
}
```

ProductPlan

```
@Override
public String toString() {
    String c2 = p2.toString();
    String[] cs2 = c2.split("\n");
    String c1 = p1.toString();
    String[] cs1 = c1.split("\n");
    StringBuilder sb = new StringBuilder();
    sb.append("->ProductPlan  (#blks=" + blocksAccessed() + ", #recs="
        + recordsOutput() + ")\n");
    // right child
    for (String child : cs2)
        sb.append("\t").append(child).append("\n");
    // left child
    for (String child : cs1)
        sb.append("\t").append(child).append("\n");
    return sb.toString();
}
```

SelectPlan

```
@Override
public String toString() {
    String c = p.toString();
    String[] cs = c.split("\n");
    StringBuilder sb = new StringBuilder();
    sb.append("->SelectPlan pred:(" + pred.toString() + ") (#blks="
        + blocksAccessed() + ", #recs=" + recordsOutput() + ")\n");
    for (String child : cs)
        sb.append("\t").append(child).append("\n");
    return sb.toString();
}
```

SortPlan

```
@Override
public String toString() {
    String c = p.toString();
    String[] cs = c.split("\n");
    StringBuilder sb = new StringBuilder();
    sb.append("->");
    sb.append("SortPlan (#blks=" + blocksAccessed() + ", #recs="
        + recordsOutput() + ")\n");
    for (String child : cs)
        sb.append("\t").append(child).append("\n");
    ;
    return sb.toString();
}
```

GroupByPlan

```
@Override
public String toString() {
    String c = sp.toString();
    String[] cs = c.split("\n");
    StringBuilder sb = new StringBuilder();
    sb.append("->");
    sb.append("GroupByPlan: (#blks=" + blocksAccessed() + ", #recs="
        + recordsOutput() + ")\n");
    for (String child : cs)
        sb.append("\t").append(child).append("\n");
    ;
    return sb.toString();
}
```

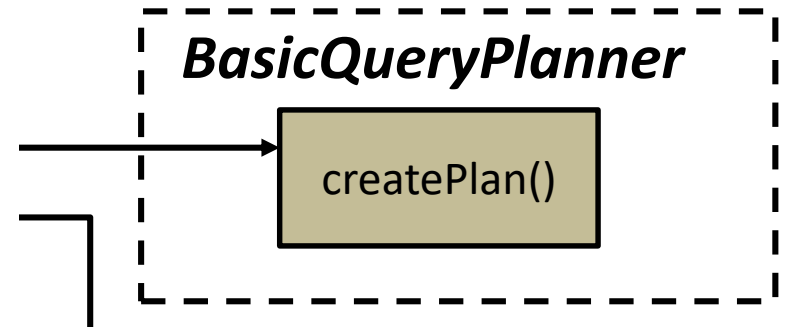
ProjectPlan

```
@Override
public String toString() {
    String c = p.toString();
    String[] cs = c.split("\n");
    StringBuilder sb = new StringBuilder();
    sb.append("->ProjectPlan (#blks=" + blocksAccessed() + ", #recs="
        + recordsOutput() + ")\n");
    for (String child : cs)
        sb.append("\t").append(child).append("\n");
    return sb.toString();
}
```

Modified/**Added** Classes

- Parse
 - *Lexer*
 - *Parser*
 - *queryData*
- Algebra
 - *ExplaiunPlan* 、 *ExplainScan*
 - *TablePlan* 、 *ProductPlan* 、 *SelectPlan* 、 *SortPlan* 、
GroupByPlan 、 *ProjectPlan*
- Planner
 - *BasicQueryPlanner*
- An example of Experiment Results

BasicQueryPlanner



BasicQueryPlanner

```
@Override
public Plan createPlan(QueryData data, Transaction tx) {
    // Step 1: Create a plan for each mentioned table or view
    List<Plan> plans = new ArrayList<Plan>();
    for (String tblname : data.tables()) {
        String viewdef = VanillaDb.catalogMgr().getViewDef(tblname, tx);
        if (viewdef != null)
            plans.add(VanillaDb.newPlanner().createQueryPlan(viewdef, tx));
        else
            plans.add(new TablePlan(tblname, tx));
    }
    // Step 2: Create the product of all table plans
    Plan p = plans.remove(0);
    for (Plan nextplan : plans)
        p = new ProductPlan(p, nextplan);
    // Step 3: Add a selection plan for the predicate
    p = new SelectPlan(p, data.pred());
    // Step 4: Add a group-by plan if specified
    if (data.groupFields() != null) {
        p = new GroupByPlan(p, data.groupFields(), data.aggregationFn(), tx);
    }
    // Step 5: Project onto the specified fields
    p = new ProjectPlan(p, data.projectFields());
    // Step 6: Add a sort plan if specified
    if (data.sortFields() != null)
        p = new SortPlan(p, data.sortFields(), data.sortDirections(), tx);
    // Step 7: Add a explain plan if the query is explain statement
    if (data.isExplain())
        p = new ExplainPlan(p);
    return p;
}
```

Modified/**Added** Classes

- Parse
 - *Lexer*
 - *Parser*
 - *queryData*
- Algebra
 - *ExplaiunPlan* 、 *ExplainScan*
 - *TablePlan* 、 *ProductPlan* 、 *SelectPlan* 、 *SortPlan* 、
GroupByPlan 、 *ProjectPlan*
- Planner
 - *BasicQueryPlanner*
- An example of Experiment Results

An Example of Experiments

- A query accessing single table with WHERE

```
SQL> EXPLAIN SELECT i_id FROM item WHERE i_id<500
```

```
query-plan
```

```
-----  
->ProjectPlan (#blks=6251, #recs=391)  
    ->SelectPlan pred:(i_id<500.0) (#blks=6251, #recs=391)  
        ->TablePlan (item) (#blks=6251, #recs=100000)
```

```
Actual #recs: 499
```

An Example of Experiments

- A query accessing multiple tables with WHERE

```
SQL> EXPLAIN SELECT d_id FROM district, warehouse WHERE d_w_id <500
```

```
query-plan
```

```
-----  
->ProjectPlan (#blks=43, #recs=40)  
  ->SelectPlan pred:(d_w_id<500.0) (#blks=43, #recs=40)  
    ->ProductPlan (#blks=43, #recs=40)  
      ->TablePlan (district) (#blks=3, #recs=20)  
      ->TablePlan (warehouse) (#blks=2, #recs=2)
```

```
Actual #recs: 40
```

An Example of Experiments

- A query with ORDER BY

```
SQL> EXPLAIN SELECT i_price FROM item ORDER BY i_price
```

```
query-plan
```

```
-----  
->SortPlan  (#blks=393, #recs=100000)  
    ->ProjectPlan  (#blks=6251, #recs=100000)  
        ->SelectPlan pred:() (#blks=6251, #recs=100000)  
            ->TablePlan (item) (#blks=6251, #recs=100000)
```

```
Actual #recs: 100000
```

An Example of Experiments

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

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

query-plan

```
-----  
->ProjectPlan (#blks=4, #recs=2)  
  ->GroupByPlan (#blks=4, #recs=2)  
    ->SortPlan (#blks=4, #recs=20)  
      ->SelectPlan pred:(d_w_id=w_id) (#blks=43, #recs=20)  
        ->ProductPlan (#blks=43, #recs=40)  
          ->TablePlan (district) (#blks=3, #recs=20)  
          ->TablePlan (warehouse) (#blks=2, #recs=2)
```

Actual #recs: 2