15721 - Project 3

Query Optimizer

Guoquan Zhao
Pedro Paredes
Qiuwen Kai
May 14, 2018
Project overview
Query Optimizer

ARCHITECTURE OVERVIEW

SQL Query

SQL Rewriter
(Optional)

Binder

Parser

Optimizer

Cost Estimates

Physical Plan

System Catalog

Annotated AST

Abstract Syntax Tree

Annotated AST
Goals

- **75% Goal:** Outer Joins;
• **75% Goal:** Outer Joins;

• **100% Goal:** Simple query rewrites;
Goals

- **75% Goal**: Outer Joins;
- **100% Goal**: Simple query rewrites;
- **125% Goal**: Expression rewrite;
Goals

• **75% Goal:** Outer Joins;
• **100% Goal:** Simple query rewrites;
• **125% Goal:** Expression rewrite;
Goals

- **75% Goal**: Outer Joins; ✓
- **100% Goal**: Simple query rewrites; ✓
- **125% Goal**: Expression rewrite;
Outer joins
Outer joins - Optimizer

Transformation Rule
Rotate Left-to-Right

Matching Plan

Implementation Rule
EQJOIN→SORTMERGE
Summary of updates

- Add unified join operator

```cpp
do add unified join operator
```

- Write outer join rules (Associativity/Commutativity)
- Fix parse tree to operator tree transform
- Add some tests
Summary of updates

- Add unified join operator
- Write outer join rules (Associativity/Commutativity)
Summary of updates

- Add unified join operator
- Write outer join rules (Associativity/Commutativity)
- Fix parse tree to operator tree transform
Summary of updates

- Add unified join operator
- Write outer join rules (Associativity/Commutativity)
- Fix parse tree to operator tree transform
- Add some tests
Query rewrites
Query rewrites - short description

SQL optimizations that don’t need a cost model

- Constant folding / propagation
Query rewrites - short description

SQL optimizations that *don’t* need a cost model

- Constant folding / propagation
- Transitive predicates
Query rewrites - short description

SQL optimizations that *don’t* need a cost model

- Constant folding / propagation
- Transitive predicates
- Trivial predicate elimination
Constant folding

```
SELECT * FROM tb1
  WHERE tb1.a = 4 * 5
```

To

```
SELECT * FROM tb1
  WHERE tb1.a = 20
```
Transitive predicates

\[
\text{SELECT } * \text{ FROM } \text{tb1, tb2}
\quad \text{WHERE } \text{tb1.a = tb2.b AND tb1.a = tb2.c}
\]

To

\[
\text{SELECT } * \text{ FROM } \text{tb1, tb2}
\quad \text{WHERE } \text{tb1.a = tb2.b AND tb1.a = tb2.c}
\quad \text{AND } \text{tb2.b = tb2.c}
\]
Trivial elimination

```
SELECT * FROM tb1
WHERE tb1.a = 5 AND tb1.b = tb1.b
```

To

```
SELECT * FROM tb1
WHERE tb1.a = 5
```
Summary of updates

- Implemented constant folding as parser step;
- Implemented transitive predicates as optimizer rule;
- Implemented trivial elimination as optimizer rule;
Summary of updates

- Implemented constant folding as parser step;
- Implemented transitive predicates as optimizer rule;
Summary of updates

- Implemented constant folding as parser step;
- Implemented transitive predicates as optimizer rule;
- Implemented trivial elimination as optimizer rule;

```
postgres=# INSERT INTO t2 VALUES(2,3,4);
INSERT 0 1
postgres=# INSERT INTO t2 VALUES(3,4,5);
INSERT 0 1
postgres=# CREATE TABLE t3(a INT);
CREATE TABLE
postgres=# INSERT INTO t3 VALUES(3);
INSERT 0 1
postgres=# INSERT INTO t3 VALUES(4);
INSERT 0 1
postgres=# INSERT INTO t3 VALUES(5);
INSERT 0 1
postgres=# SELECT * FROM t3 LEFT OUTER JOIN (t1 INNER JOIN t2 ON t1.a=t2.d) ON t3.d = t2.d WHERE t2.c = 2;
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
(3 rows)
postgres=# explain SELECT * FROM t3 LEFT OUTER JOIN (t1 INNER JOIN t2 ON t1.a=t2.d) ON t3.d = t2.d WHERE t2.c = 2;
Query plan
```
```
Thanks!