SQL Parser: From MiniOB to Real-World

Ziqin Li
System Research Association@Sichuan University

October 21, 2024

DB (Frontend) is a Compiler

- Lexer
- Parser
- Rewriter
- Optimizer

Logical Operators are IR.

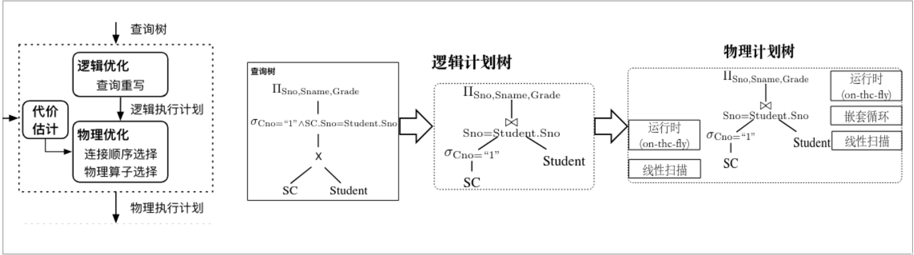


Figure: Processing SQL statement. Ref: Guoliang Li's slide

MiniOB

What did we have?

Abstract class:

- class Value
- class Expression
- class Condition

Parser & Lexer:

- A Perfect Lexer (in fact, lexers are easy to be perfect)
- Basic Parser Capabilities

Rewriter and Optimizer

- No Optimizer
- Rewriter Supporting Some Rules

MiniOB
What didn't we have?

- Expressions in Conditions
- Simple nor Complex Sub Query
- NULL
- Join Tables On
- Alias
- Create Table by Select
- *Plan Cache
- *Query Cache

and so on...

MiniOB

Logical Operators

```
enum class LogicalOperatorType
{
  CALC,
  TABLE_GET,    ///< 从表中获取数据
  PREDICATE,    ///< 过滤，就是谓词
  PROJECTION,   ///< 投影，就是select
  JOIN,         ///< 连接
  INSERT,       ///< 插入
  DELETE,       ///< 删除，删除可能会有子查询
  EXPLAIN,      ///< 查看执行计划
  UPDATE,       ///< 修改
  GROUP_BY,     ///< 分组
};
```

MiniOB

Relational Algebra's Operators

- Set Operator
  - Union
  - Difference
- Projection
- Selection
- Join (Natural Join, Semi Join ...)

More: Calc, TABLE_GET, EXPLAIN, UPDATE, GROUP_BY

MiniOB

Parser and Lexer

```
condition:
    rel_attr comp_op value
    {
      $$ = new ConditionSqlNode;
      $$->left_is_attr = 1;
      $$->left_attr = *$1;
      $$->right_is_attr = 0;
      $$->right_value = *$3;
      $$->comp = $2;

      delete $1;
      delete $3;
    } et al
```

MiniOB

Parser and Lexer

Must the Value be ”the Value” ?

Change `ConditionSqlNode`, but how will the executor execute this SQL statement, or how they will leverage the `ConditionSqlNode`.

ConditionSqlNode -> FilterStmt -> PredicateLogicalOperator -> PredicatePhysicalOperator.

Volcano model for execute a expression.

Volcano - An Extensible and Parallel Query Evaluation System TKDE’94

Interlude: Volcano Model

Calc a Expression

$$100 \times \alpha + 5\frac{\beta}{\alpha} + 30 \times \beta$$

How to calc it when knowing α and β?

Interlude: Volcano Model

Calc a Expression

$$100 \times \alpha + 5\frac{\beta}{\alpha} + 30 \times \beta$$

How to calc it when knowing α and β?
How to calc when we don't know it?

Interlude: Volcano Model

Calc a Expression

$$100 \times \alpha + 5\frac{\beta}{\alpha} + 30 \times \beta$$

How to calc it when knowing α and β?
How to calc when we don't know it?
How to calc when we don't know it and have a bunch of α and β to calc?

Interlude: Volcano Model

Calc a Expression

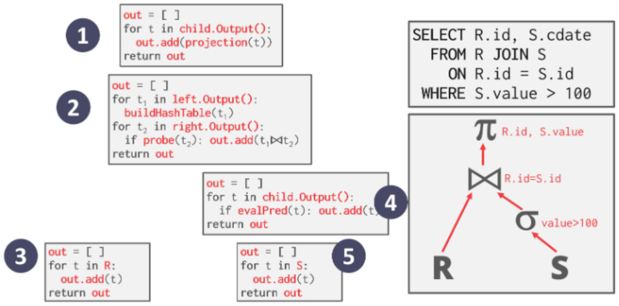
$$100 \times \alpha + 5\frac{\beta}{\alpha} + 30 \times \beta$$

How to calc it when knowing α and β?
How to calc when we don't know it?
How to calc when we don't know it and have a bunch of α and β to calc?

- Volcano Model
- Vector Model
- Materialization Model

Interlude: Volcano Model

Materialization Model



```
out = [ ]
for t in child.Output():
  out.add(projection(t))
return out
```

```
out = [ ]
for t1 in left.Output():
  buildHashTable(t1)
for t2 in right.Output():
  if probe(t2): out.add(t1⋈t2)
return out
```

```
out = [ ]
for t in child.Output():
  if evalPred(t): out.add(t)
return out
```

```
out = [ ]
for t in R:
  out.add(t)
return out
```

```
out = [ ]
for t in S:
  out.add(t)
return out
```

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Figure 2: Materialization Model Example – Starting at the root, the `child.Output()` function is called, which invokes the operators below, which returns all tuples back up.

Figure:

Interlude: Volcano Model

Iterator Model

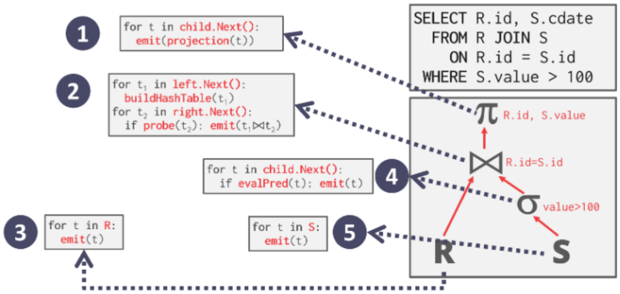
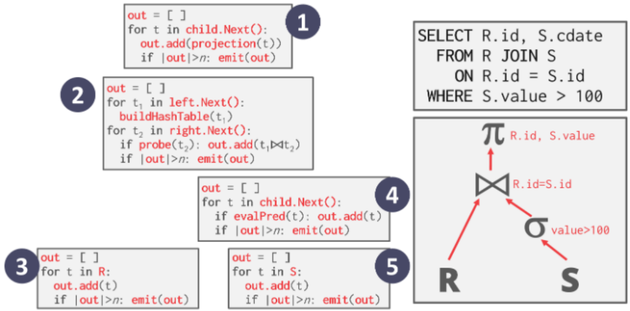


Figure 1: Iterator Model Example – Pseudo code of the different `Next` functions for each of the operators. The `Next` functions are essentially `for`-loops that iterate over the output of their child operator. For example, the root node calls `Next` on its child, the join operator, which is an access method that loops over the relation R and emits a tuple up that is then operated on. After all tuples have been processed, a null pointer (or another indicator) is sent that lets the parent nodes know to move on.

Interlude: Volcano Model

Vector Model



```
out = [ ]
for t in child.Next():
  out.add(projection(t))
  if |out|>n: emit(out)
```

```
out = [ ]
for t1 in left.Next():
  buildHashTable(t1)
for t2 in right.Next():
  if probe(t2): out.add(t1⋈t2)
  if |out|>n: emit(out)
```

```
out = [ ]
for t in R:
  out.add(t)
  if |out|>n: emit(out)
```

```
out = [ ]
for t in child.Next():
  if evalPred(t): out.add(t)
  if |out|>n: emit(out)
```

```
out = [ ]
for t in S:
  out.add(t)
  if |out|>n: emit(out)
```

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Figure 3: Vectorization Model Example – The vectorization model is very similar to the iterator model except at every operator, an output buffer is compared to the desired emission size. If the buffer is larger, then a tuple batch is sent up.

Figure:

Table Join on

```
rel_list:
    relation {
      $$ = new std::vector<std::string>();
      $$->push_back($1);
      free($1);
    }
    | relation COMMA rel_list {
      if ($3 != nullptr) {
        $$ = $3;
      } else {
        $$ = new std::vector<std::string>;
      }

      $$->insert($$->begin(), $1);
      free($1);
    }
```

Table Join On

Implementation

```
join: join | inner join
rel_join: relation join relation | relation join relation ON c
rel_list: relation | relation COMMA rel_list | relati
```

Real-world SQL Parser

MySQL's Parser

sql/sql_yacc.yy

- 18K LoC
- View

Apache Calcite

Real-world SQL Parser

Optimizer

Logical: rewrite (predicate pushdown) Physical: A join B: A, B have index.