The Volcano Optimizer Generator: Extensibility and Efficient Search

Presenter: Zicun Cong

School of Computing Science
Simon Fraser University

CMPT843, February 6, 2016
Outline

Introduction

Method

Experiment

Conclusion
Query Execution

- **Logical Operator**: A function map the operator’s inputs to its outputs. (e.g. join, selection, projection)

- **Physical Operator**: An algorithm that implements a logical operator. (e.g. hash join, merge join)

- **Operator Expression**: A hierarchy of operators.

- **Execution Plan**: An expression made up entirely of physical operators

![Logical Expression](Diagram1.png)

- **Logic**
  - **Operator**
  - **A**
  - **B**
  - **C**

![Physical Expression](Diagram2.png)

- **Physical**
  - **Operator**
  - **INDEX_SCAN(A)**
  - **FILE_SCAN(B)**
  - **FILE_SCAN(C)**
  - **Merge Join**
Query Optimization

- **Definition:** For a given query, find an execution plan for it that has the lowest cost
- **Challenges:** The number of candidate execution plans is huge
  - Equivalent expressions (e.g. $A \bowtie (B \bowtie C) = (A \bowtie B) \bowtie C$)
  - Different algorithms for each operation (e.g. Sort Merge Join, Hash Join, Nested Loop Join)
Motivation

Cost difference between evaluation plans for a query can be enormous (e.g. seconds vs. days)

Existing Methods:

- Heuristic search: Use static rules to generate plan
- Stratified search: Planning is done in multiple stages
- Unified search: Perform query planning all at once

Problems:

- Not effective: Hard to generate good plans for complex queries
- Not extensible: Rules maintenance is a huge pain
- Not efficient: Waste a lot of effort in searching
Overview of Volcano

Objectives

- Usability as a stand-alone tool
- More efficient resource usage
- Extensible support for physical properties
Optimizer Components

**Physical property:** The properties of physical layout of data

**Logical Rules:** Logic-Logic transformation

**Implementation Rules:** Logic-Implementation transformation

**Algorithm/Enforcer:** Have required physical properties of their inputs, physical properties of its output, and cost function

**Example**

- Physical property: Select * FROM A ORDER BY id
- Logic Rules: $S \bowtie R = R \bowtie S$
- Implementation Rules: $\bowtie \rightarrow \{\text{Merge Join, Hash Join, Sort + Hash Join}\}$
- Merge Join requires the inputs to be sorted on the join attributes and produces the output sorted
Optimizer Generator

Model Specification

- A set of logical operators and algebraic transformation rules
- A set of algorithms, enforcers and implementation rules
- An applicability function for each algorithm and enforcer
- A cost function for each algorithm and enforcer
- Property function for each operator, algorithm and enforcer

Generate Optimizer
Plan Search Engine

Basic Ideas

- The optimal execution plan of a query is composed of the optimal execution plan of its sub-queries.
- If the cost of a sub execution plan $P$ is larger than cost threshold, any larger execution plan containing $P$ can be pruned.
- Recursively divide the query into sub-queries and find optimal execution plans of each sub-queries.

Example

SELECT * FROM A, B, C WHERE A.id = B.id AND B.id = C.id

Cost($A \bowtie (B \bowtie C)) = Cost(A) + Cost(\bowtie) + Cost(B \bowtie C)$
Plan Search Engine

Enumerate Search Space

Three possible moves

- Logic-Logic Transform
- Logic-Implementation Transform
- Enforcers for required physical property

Example

SELECT * FROM R, S, T WHERE R.x=S.x AND S.y=T.y
ORDER BY S.y
Algorithm Overview

FindBestPlan (LogExpr, PhysProp, Limit)

/*Check look-up table (LogExpr, PhysProp) -> (Execution Plan, Cost)*/
If (LogExpr, PhysProp) in table
  /*Prune high cost plans*/
  If cost < Limit return (Execution Plan, Cost)
  else return Fail
/* else: optimization required */
Generate possible moves
For move in moves
  Handle the move
/* maintain the look-up table of explored facts */
If LogExpr is not in the look-up table:
  Update loop-up table
return best Plan and Cost
Handle Moves

Handle Logical Transformation

Recursively call the **FindBestPlan** function with the new logical expression

**Examples**

```
FindBestPlan(((R⋈S)⋈T), S.y, 500)

Logic-Logic:
(R⋈S)⋈T)=R⋈(S⋈T)
Cost=0

FindBestPlan( R⋈(S⋈T), S.y, 500)
```
Handle Moves

Handle Implementation Transformation

1. Update the total cost. $\text{TotalCost} = \text{Algorithm Cost}$
2. Recursively call $\text{FindBestPlan}$ on each input of the algorithm

Examples

$$\text{FindBestPlan}((R \bowtie S) \bowtie T), S.y, 500)$$

Logic-Implementation:

$(R \bowtie S)$ Merge Join T
Cost=100

$$\text{FindBestPlan}((R \bowtie S), S.y, 400)$$
$$\text{FindBestPlan}(T, S.y, 400)$$
Handle Moves

Handle Enforcer Transformation
1. Update physical property and cost
2. Recursively call the FindBestPlan function with the new physical property and cost

Examples

```
FindBestPlan(((R<R>S)>T), S.y, 500)
```

Enforcer:
```
Sort(((R<R>S)>T)) by S.y
Cost=200
```

```
FindBestPlan( (R<R>S)>T), null, 300)
```
Depth-First Search Tree

Query

SELECT * FROM R, S, T WHERE R.x=S.x AND S.y=T.y
ORDER BY S.y
Experiment
Discussion

Pros

▶ Unified, fully cost-based model
▶ Easily add new operations and equivalence rules
▶ Pruning strategies can effectively reduce the searching space

Cons

▶ Only compare their method with one query optimization method
▶ Exhaustively generates all logically equivalent expressions
Conclusion

- Volcano improves the work of EXODUS
- Top-down optimizer, uses dynamic programming and memoisation
- Enumerates physical search space in depth-first order
- Uses branch and bound pruning to prune the search space
- Use physical properties to direct the search
- Volcano has gained widespread acceptance in the industry as a state-of-the-art optimizer; the optimizers of Microsoft SQL Server [21] and Tandem ServerWare SQL Product [6] are based on Volcano
Backup Slids
Logical / Implementation Rules

- **Logical Rules:** The algebraic rules of expression equivalence. They are used to get equivalent logical expressions.

- **Implementation Rules:** The possible mappings of operators to algorithms

- Algorithms have three properties
  - Required physical properties of its inputs
  - Physical properties of its output
  - Cost function

Example

- Commutative law: $R \Join S = S \Join R$
- Join can be implemented by merge-join, hash join or nested-loop join.
- merge-join requires the inputs to be sorted on the join attributes and produces the output sorted
Optimizer Generator

Model Specification

- A set of logical operators and algebraic transformation rules
- A set of algorithms, enforcers and implementation rules
- An applicability function for each algorithm and enforcer
- A cost function for each algorithm and enforcer
- Property function for each operator, algorithm and enforcer

- Given a model specification, Volcano generates a query optimizer.
- Optimizer source code is compiled and linked with the other DBMS software such as the query execution engine and with the search engine.
- When the DBMS is operational, the input query is passed to the optimizer, which generates an optimized plan for it.
Difference between EXODUOS

- Volcano took less time to optimize
- EXODUS optimizer generator measurements were quite volatile and took a lot of memory
- EXODUSs generated optimizer and search engine do not explore and exploit physical properties
Difference between Starburst

- Volcano took less time to optimize
- New operators are integrated at the query rewrite level optimization in this level is heuristic
- Starburst has a hierarchy of intermediate levels
Physical Transformation

PQDAG for A $\bowtie$ B on A.X = B.Y

dotted boxes - logical eqv. nodes; solid boxes - phys. eqv. nodes
Algorithm

FindBestPlan (LogExpr, PhysProp, Limit)
• if the pair LogExpr and PhysProp is in the look-up table
  • if the cost in the look-up table < Limit
    • return Plan and Cost
  • else return failure
• /* else: optimization required */
• create the set of possible "moves" from
  • applicable transformations
  • algorithms that give the required PhysProp
  • enforcers for required PhysProp
• order the set of moves by promise
Algorithm

• for the most promising moves
  • if the move uses a transformation
    • apply the transformation creating NewLogExpr
    • call FindBestPlan (NewLogExpr, PhysProp, Limit)
  • else if the move uses an algorithm
    • TotalCost := cost of the algorithm
    • for each input I while TotalCost < Limit
      • determine required physical properties PP for I
        • Cost = FindBestPlan (I, PP, Limit – TotalCost)
        • add Cost to TotalCost
  • else /* move uses an enforcer */
    • TotalCost := cost of the enforcer
    • modify PhysProp for enforced property
    • call FindBestPlan for LogExpr with new PhysProp
Algorithm

- /* maintain the look-up table of explored facts */
- if LogExpr is not in the look-up table
  - insert LogExpr into the look-up table
  - insert PhysProp and best plan found into look-up table
- return best Plan and Cost