The Volcano Optimizer Generator:

Extensibility and Efficient Search

Presented By : Ravi Bisla(301345992)

About the Paper

Publishing Details:

- April 19 - 23, 1993
- Published in proceedings of the Ninth International Conference on Data Engineering.

Authors:

- William J. McKenna
- Goetz Graefe(Also the author EXODUS optimizer generation)

Optimizer Generators

Query optimizer : takes query in a logical algebraic expression and output a plan to efficiently execute the query.

Optimizer generators : take in a model of queries and plans and output a query optimizer that performs the actual optimization.

Question: how a query is converted from **raw form into logical alberic expression**?

Query Optimizer Workflow

Query Optimizer :

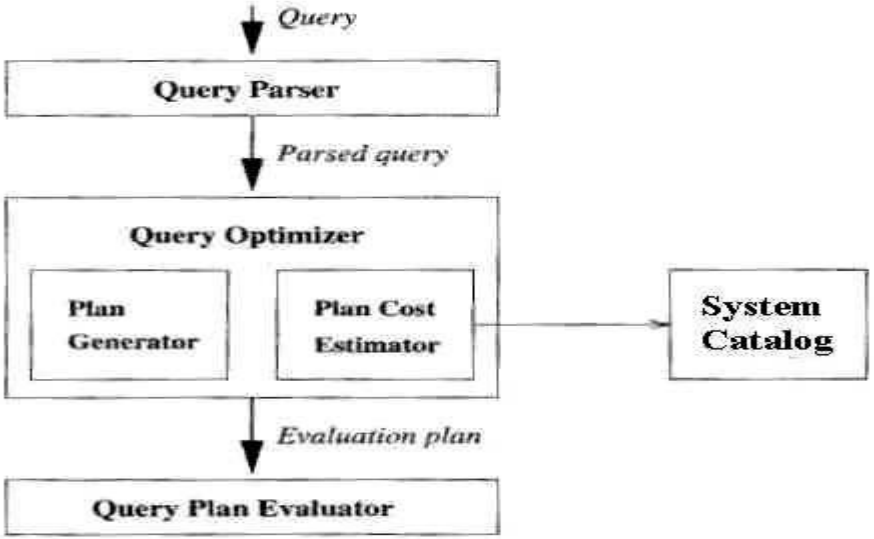


Figure 1

Volcano Optimizer Generator

It is an improved version of the authors' previous work on the EXODUS optimizer generator.

Two important features :

- Efficient search algorithm.
- Extensibility

Volcano Optimizer Generator

From previous experiments with EXODUS it was concluded that performance should not be sacrificed for extensibility for two main reasons :

- Data Volume Continues to grow
- To overcome acceptance problem Database systems must achieve at least the same performance as file system in use.

EXODUS Optimizer Generator

- Earlier optimizer generator prototype, studied before designing VOLCANO optimizer generator

- Learnings from EXODUS:
  - Proven the feasibility and validity of the optimizer generator paradigm
  - Difficult to construct efficient production-quality optimizers

The Outside View of Volcano

Volcano provides a **framework for database implementers** to input a model specifying the properties of their physical and logical model.

For each user query, the database implementer passes that query to the Volcano optimizer generator, and Volcano outputs an optimized execution plan for that query.

Optimizer Generator Input and Optimizer Output

Volcano's job is to map an expression in the logical algebra into an expression in the physical algebra using defined rules.

Logical algebra - describes a user's query

Physical algebra - describes a query evaluation plan consisting of algorithms.

Optimizer Generator Input and Optimizer Output

This section describes components that the **optimizer implementer** defines when implementing a new database query optimizer :

1)A set of logical operators : that compose the logical algebra in which queries are written. For example, the logical algebra of a relational database is the relational algebra, and the operators include things like **select, project, and join**.

Optimizer Generator Input and Optimizer Output

2) A set of physical algorithms : that compose the physical algebra in which plans are formed. For example, the physical operators of a relational database would be things like sort-merge join, hash join, nested loop join, full table scan, etc.

3) A set of algebraic transformation rules which transform a logical algebra expression into another logical algebra expression.

For example, **a JOIN b -> b JOIN a** is a rule which says joining is commutative.

Optimizer Generator Input and Optimizer Output

4) A set of implementation rules which define how a logical algebra expression can be mapped to a physical algebra expression.

For example, **a JOIN b -> a TNLJ b** is a rule which says that a join can be implemented using a tuple-nested loop join

5) **Enforcers :** Some algorithms have no corresponding logical algebra operator. For example, the sort algorithm has no corresponding logical relational operator. These algorithms are called enforcers and are used to enforce certain physical properties without affecting logical semantics.

Optimizer Generator Input and Optimizer Output

6) An abstract cost function for each algorithm and enforcer. The cost function assigns a cost to each algorithm and enforcer. Volcano optimizer try to find a plan with minimum cost, but the definition of cost is left to the database implementor.

7) An abstract data type for logical and physical properties. Logical properties describe logical algebra expressions and Physical properties describe physical algebra expressions. For example, a logical property might be the number of output tuples or the schema of the output and a physical property might be the sort order.

Optimizer Generator Input and Optimizer Output

8) An applicability function for each algorithm and enforcer which returns whether or not an algorithm or enforcer can implement a logical algebra expression and provide a certain set of physical properties.

Design Principles

Five Design Principles

1st Principle : Volcano allows for an extensible algebraic set of operators that describe the operation of the underlying database in terms of equivalence relationships,and implementation algorithms.

Design Principles

2nd Principle : Use concept of rule

- Focus on independent rules make the design more modular.

- Rules allow database implementer’s to describe general algebraic relationships.

Design Principles

3rd Principle : Don’t use intermediate representations.

- Straight from queries to plans.
- It simplifies the life of database implementer as they no longer have to specify a intermediate representations.

4th Principle: Compile plans, don't interpret them.

- Volcano made the choice to generate compiled code, rather than code that may be interpreted.
- Interpretation can be more flexible but compiled rule typically executes faster

Design Principles

5th Principle: Dynamic Programming in the search algorithm

- It uses dynamic programming by retaining a large set of partially evaluated results and using these results in later decisions.

The Search Engine

- Search engine and it’s algorithm are central components.

- Uses dynamic programming and is very goal oriented - driven by needs rather than by possibilities

The Search Engine

The algorithm takes as input :

- A logical expression to optimize

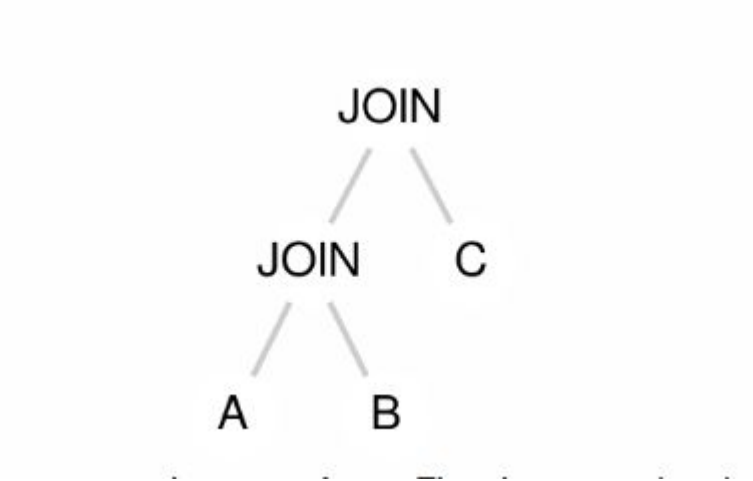
- A set of physical properties to satisfy (e.g. a sort order specified by a query)

- A cost limit. The cost limit can be set to infinity, but can also be set to some smaller number to ensure that no outrageously expensive plan is generated.

The Search Engine

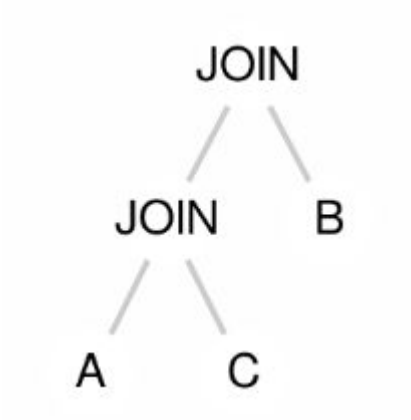
Algorithm can make following move :

Example:



The Search Engine

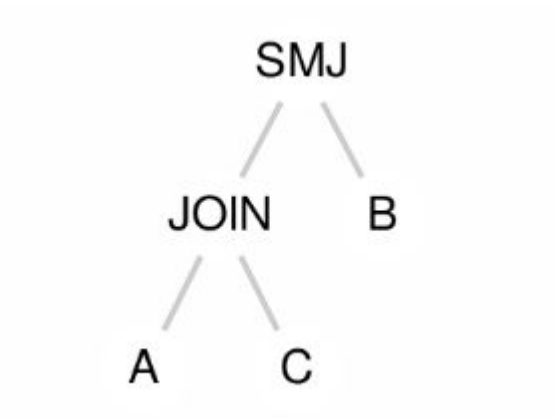
Move 1: It can apply a logical transformation rule to convert the query to an equivalent query. For example, we might rewrite the query to reorder the join, as shown below. After we transform the query, we recurse.



The Search Engine

Move 2: It can instantiate a logical operator with a physical algorithm.

We then recurse on the children of the algorithm using the required physical properties generated by the applicability function. **If we ever bust the cost limit, we bail out early**. Here, we instantiate the top join with a sort merge join



Comparison to the EXODUS Optimizer Generator

- EXODUS did not distinguish logical from physical algebra which led to inefficiencies.
- EXODUS had no notion of physical properties.
- EXODUS did not use a top-down search strategy and ended up re-analyzing a lot of things.
- EXODUS did not have a generic cost function.

Key Takeaways

Extensibility was achieved by generating optimizer source code from data model specifications and by encapsulating costs as well as logical and physical properties into abstract data types

Efficiency was achieved by using dynamic programming, and by retaining a large set of partially evaluated results and using these results in later decisions.

In effect, using a backward chaining to search through a decision tree for the most optimal query plan.

THANK YOU!

Any questions?