presented at the Texas Instruments Workshop on Query Optimizers, 1993

Prairie: An Algebraic Framework for Rule Specification in Query Optimizers

Dinesh Das
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

September 13, 1993

Overview of talk

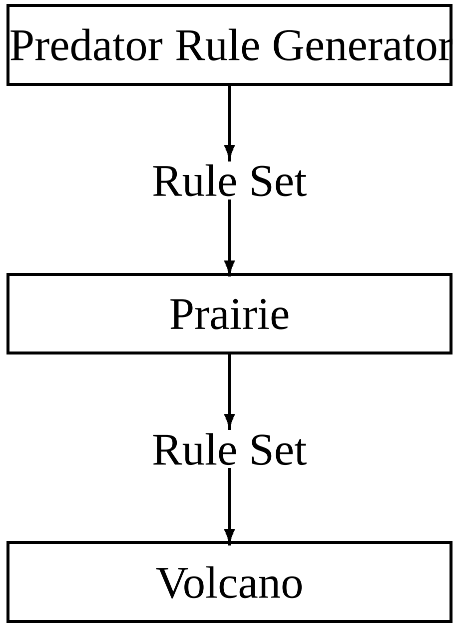
What is the problem?

- Rule-based query optimization.
- Develop a simple, well-defined framework called Prairie to specify rules in a rule-based query optimizer.
  - Abstraction of rule specification to insulate user from rule engine.
  - Support automatic generation of rule sets.
- Test framework by using it to write query optimizers.
- Develop a pre-processor to translate Prairie rule specification to Volcano.
- Test performance of Prairie vs. Volcano.

Why is it important?

- A simple framework makes it easy to read, write and understand rules with fewer errors.
- Need to distill the essence of what needs to be specified by the user.
  - Too many details to specify.
  - Software Engineering approach.
  - Research based on Volcano.

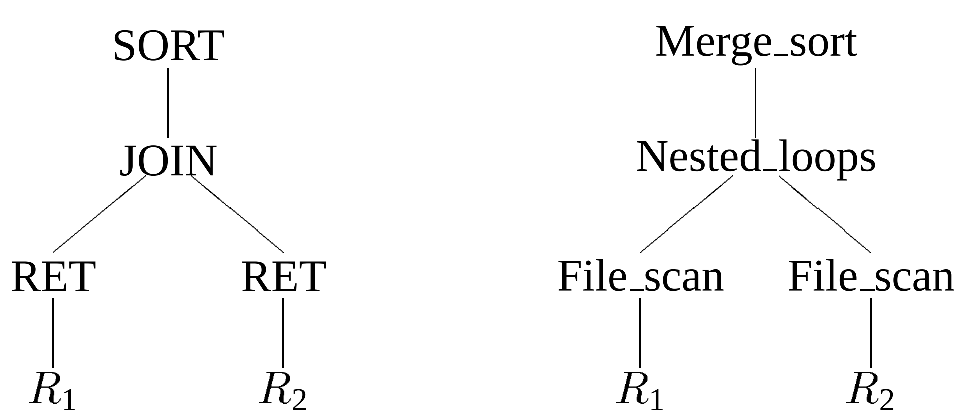
Motivation



- Abstract specification of rules – only essential things to be specified.
- Minimize complexity (i.e., dependence on implementation details) of rule set specification.
- Admits different types of rule set optimizations:
  - Collapsing of rule sets.
  - Generate implementation details.
- Allows automatic generation of rule sets to be easier.

Prairie: Terminology

- **Stored files and streams.** Stored files are relations. Streams are produced by computations on stored files or other streams.
- **Operators.** Abstract computations on streams or files. Describes what the *semantics* are, not *how* they are implemented.
- **Algorithms.** Concrete implementations of operators.
- **Operator Tree.** Rooted tree with operators or algorithms as internal nodes, stored files as leaves.
- **Access Plan.** Operator tree with algorithms as internal nodes.
- **Property.** Information used by optimizer to choose between different operator trees/access plans.



Prairie: Terminology

- **Descriptors.** List of ⟨*property,value*⟩ pairs that encode operator tree node information.

Example:

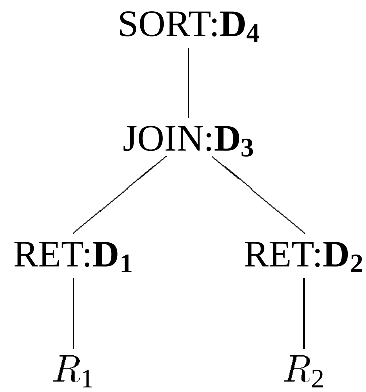
Property	**Meaning**
join_predicate	join predicate for JOIN operator
tuple_order	tuple order of stream
num_records	number of tuples in stream
tuple_size	size of individual tuple in stream
attribute_list	list of attributes
cost	estimated cost of algorithm

- Lot of effort in coding properties. We ultimately may want to provide with Prairie a library of pre-written implementations of certain properties.

Prairie: Terminology

- **Expression.** Operator tree with descriptor information attached.

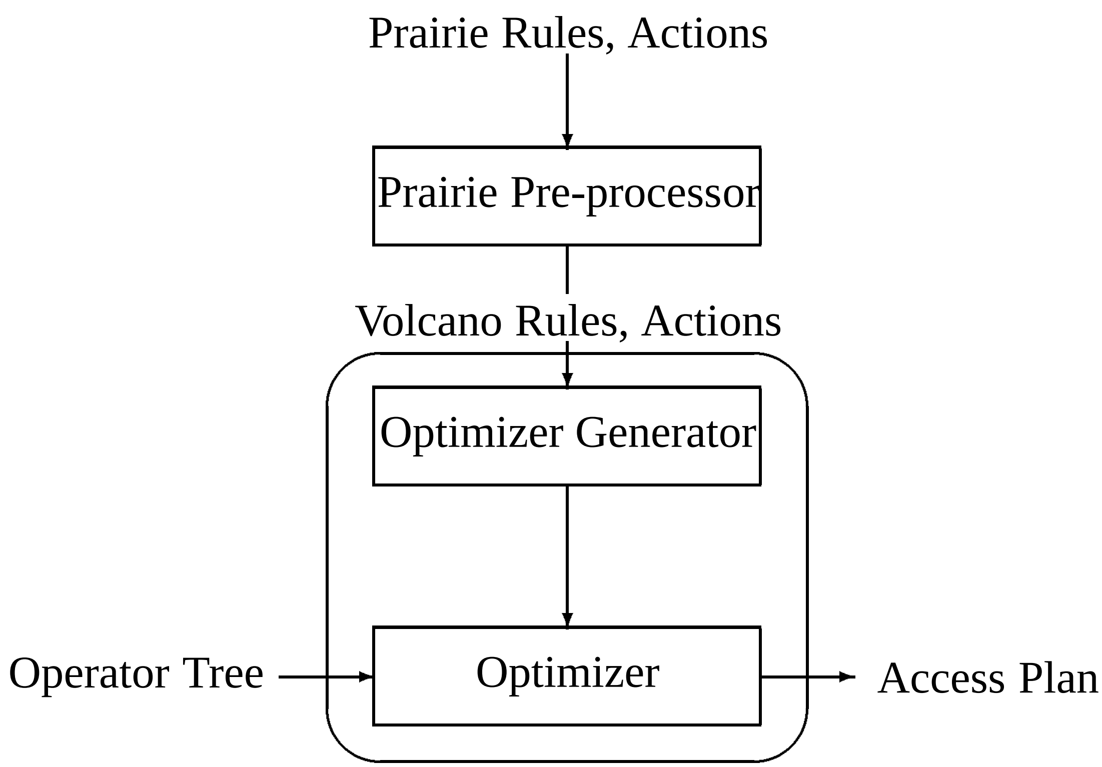
SORT(JOIN(RET(R_1):$\mathbf{D_1}$, RET(R_2):$\mathbf{D_2}$):$\mathbf{D_3}$):$\mathbf{D_4}$



Descriptor members are accessed by a structure member relationship. Thus,
$\mathbf{D_2}$.num_records = # of records returned by RET(R_2).
$\mathbf{D_4}$.num_records = # of records returned by the entire expression.

- **Rules.** (Largely influenced by Volcano)
  Rules transform one operator tree into another. There are two types of rewrite rules: T-rules ("transformation rules") and I-rules ("implementation rules"). Each type has a test and actions associated with it.

Prairie: Model



- Actions written in C.
- Top-down optimization
- Goal is to generate code with comparable efficiency to a hand-written version, yet input is much simpler.

Prairie: T-Rules

General Form: (influenced by Volcano)

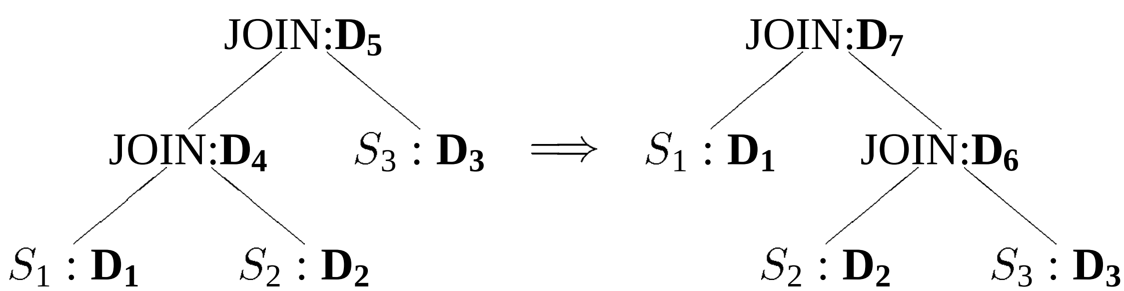
$$E(x_1, \ldots, x_n) : \mathbf{D_1} \Longrightarrow E'(x_1, \ldots, x_n) : \mathbf{D_2}$$

$$\{\{ \text{ pre-test statements } \}\}$$

$$\text{test}$$

$$\{\{ \text{ post-test statements } \}\}$$

Example: JOIN Associativity



$$\text{JOIN}(\text{JOIN}(S_1, S_2) : \mathbf{D_4}, S_3) : \mathbf{D_5} \Longrightarrow \text{JOIN}(S_1, \text{JOIN}(S_2, S_3) : \mathbf{D_6}) : \mathbf{D_7}$$

{{

}}

! attr_in_attr_list ($\mathbf{D_5}$.join_predicate.operand1, $\mathbf{D_2}$.attributes)

{{

$\mathbf{D_7}$ = $\mathbf{D_5}$;

$\mathbf{D_7}$.join_predicate = $\mathbf{D_4}$.join_predicate;

$\mathbf{D_6}$.attributes = union($\mathbf{D_2}$.attributes, $\mathbf{D_3}$.attributes);

$\mathbf{D_6}$.join_predicate = $\mathbf{D_5}$.join_predicate;

$\mathbf{D_6}$.tuple_size = estimate_tuple_size($\mathbf{D_2}$.tuple_size, $\mathbf{D_3}$.tuple_size;)

$\mathbf{D_6}$.num_records = estimate_num_records($\mathbf{D_2}$.num_records, $\mathbf{D_3}$.num_records);

}}

Prairie: I-Rules

General Form: (influenced by Volcano)

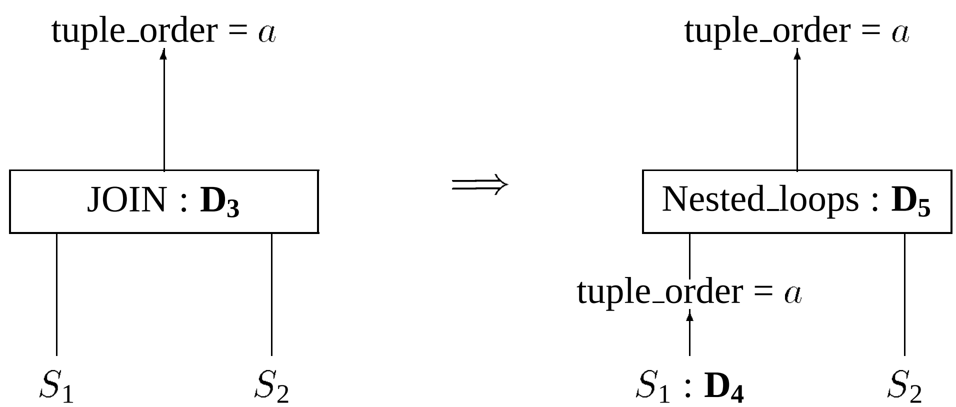
$$E(x_1, \ldots, x_n) : \mathbf{D_1} \Longrightarrow A(x_1, \ldots, x_n) : \mathbf{D_2}$$

test

$\{\{$ pre-opt statements $\}\}$

$\{\{$ post-opt statements $\}\}$

Example: Nested Loops



$$\text{JOIN}(S_1, S_2) : \mathbf{D_3} \Longrightarrow \text{Nested\_loops}(S_1 : \mathbf{D_4}, S_2) : \mathbf{D_5}$$

TRUE

$\{\{$

$$\mathbf{D_5} = \mathbf{D_3};$$

$$\mathbf{D_4} = \mathbf{D_1};$$

$$\mathbf{D_4}.\text{tuple\_order} = \mathbf{D_3}.\text{tuple\_order};$$

$\}\}$

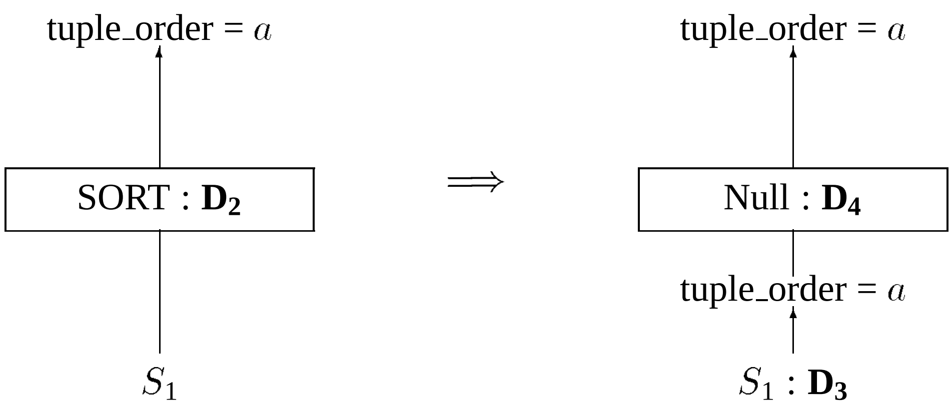
$\{\{$

$$\mathbf{D_5}.\text{cost} = \mathbf{D_4}.\text{cost} + (\mathbf{D_4}.\text{num\_records}) * \mathbf{D_2}.\text{cost};$$

$\}\}$

Prairie: Null Algorithm

The Null algorithm serves to pass constraints of a node down to its input. Since operators are *explicit* in Prairie, we need a way to remove operators as necessary from operator trees.



Example: Null SORT

$$\text{SORT}(S_1) : \mathbf{D_2} \Longrightarrow \text{Null}(S_1 : \mathbf{D_3}) : \mathbf{D_4}$$

$$\text{TRUE}$$

$$\{\{$$

$$\begin{aligned} \mathbf{D_4} &= \mathbf{D_2}; \\ \mathbf{D_3} &= \mathbf{D_1}; \\ \mathbf{D_3}.\text{tuple\_order} &= \mathbf{D_2}.\text{tuple\_order}; \end{aligned}$$

$$\}\}$$

$$\{\{$$

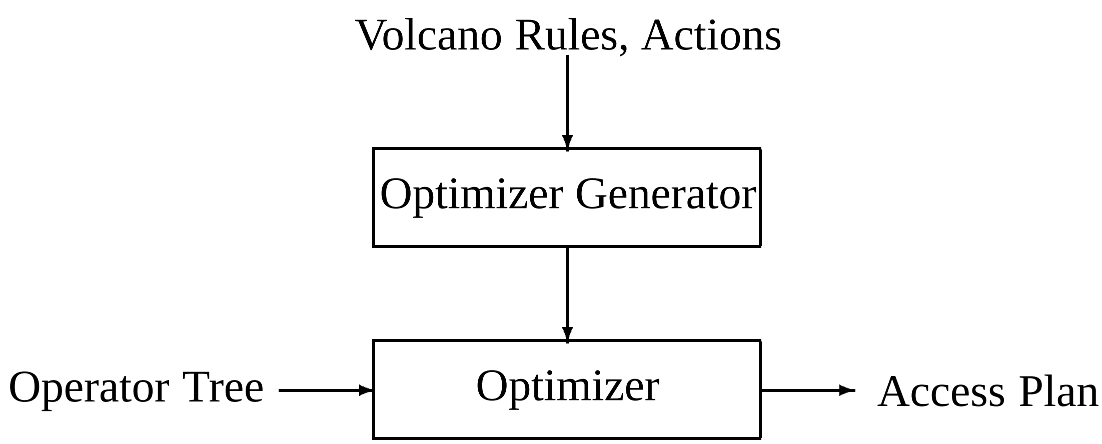
$$\mathbf{D_4}.\text{cost} = \mathbf{D_3}.\text{cost};$$

$$\}\}$$

Volcano: Model

Volcano Model (Graefe, 1990)

- Top-down query optimization.



- Extensibility — can add new operators, algorithms.
- Two types of rules:
  - Transformation rules (non-cost-based rewrites).
  - Implementation rules (cost-based rewrites).
- Efficient storage of equivalence classes of operator trees.
- Branch-and-bound control of search space.
- Constraint-based generation of "interesting expressions" (ala System R).

Volcano: Model

- "Hidden" algorithms, rules: enforcers.

Why do we care?

- Hard to visualize where enforcers will appear in operator tree since enforcers don't appear in rules.
- Hard to see how operator trees are rewritten. Also, greater potential for errors in specifying rewrite rules.

Conclusion: Make all operators and algorithms *explicit*.

Volcano: Model

- Operator Tree node information represented using five different structures:
  - Operator/Algorithm arguments
  - Logical property
  - System property
  - Physical property
  - Cost
- Adding new operators/algorithms may require repartition of properties.
  Example. Adding a relational project operator changes attribute_list from a logical to physical property.

Volcano: Model

- Requires writing of functions to map properties between operator trees.
  - derive_log_prop.
  - derive_sys_prop.
  - derive_phy_prop.
  - do_any_good.
  - get_input_pv.
  - cost.

  Conclusion: Use *one* structure to encode node information.

- Volcano implementation rules hard to read, write, understand because of scattered functions.
  - easy to make mistakes.

The Problem

- Translating Prairie input to Volcano.
  - Automatic mapping of descriptor to different vectors in Volcano.
  - Automatically generate mapping functions in Volcano.

Preliminary Experimental Results

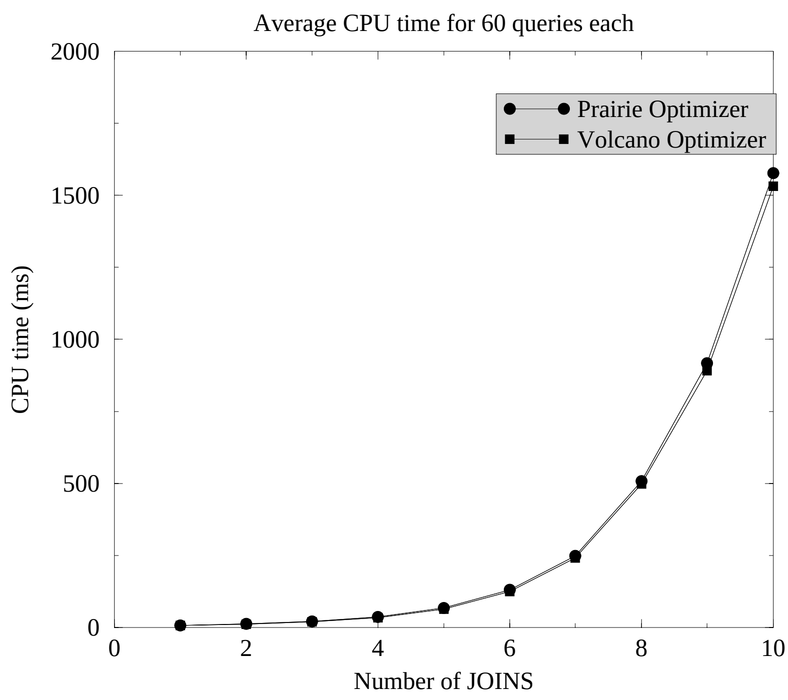
Specification of a simple centralized optimizer:

- JOIN, RET, SORT operators.
- One JOIN algorithm, Jmerge; one RET algorithm, File_scan.
- One SORT algorithm, Merge_sort.

Lines of code

Prairie	Volcano from P2V	Hand-written Volcano
700	1300	1400

n-way JOIN queries



Other experiments: Multiple JOIN algorithms (Jmerge, Jhash), collapsing of multiple rules.

Future Work

- Encode TI's OODB optimizer to see if Prairie scales.
- Investigate extending Prairie to build layered query optimizers.
  - Each layer consists of a set of rules.
  - Layers can be composed in arbitrary ways.
  - Rules in different layers can be composed for efficiency.
- May investigate extending Prairie to specify bottom-up query optimizers.