Query Optimization

Instructor: Matei Zaharia
Query Execution Overview

Query representation (e.g. SQL) → Logical query plan (e.g. relational algebra) → Optimized logical plan → Physical plan (code/operators to run)
Outline

What can we optimize?

Rule-based optimization

Data statistics

Cost models

Cost-based plan selection
Outline

What can we optimize?

Rule-based optimization

Data statistics

Cost models

Cost-based plan selection
What Can We Optimize?

**Operator graph:** what operators do we run, and in what order?

**Operator implementation:** for operators with several impls (e.g. join), which one to use?

**Access paths:** how to read each table?
  » Index scan, table scan, C-store projections, ...
Typical Challenge

There is an exponentially large set of possible query plans

- Access paths for table 1
- Access paths for table 2
- Algorithms for join 1
- Algorithms for join 2
- ...

Result: we’ll need techniques to prune the search space and complexity involved
Outline

What can we optimize?

Rule-based optimization

Data statistics

Cost models

Cost-based plan selection
What is a Rule?

Procedure to replace part of the query plan based on a pattern seen in the plan

**Example:** When I see $expr \text{ OR TRUE}$ for an expression $expr$, replace this with $\text{TRUE}$
Implementing Rules

Each rule is typically a function that walks through query plan to search for its pattern

```java
void replaceOrTrue(Plan plan) {
    for (node in plan.nodes) {
        if (node instanceof Or) {
            if (node.right == Literal(true)) {
                plan.replace(node, Literal(true));
                break;
            }
            // Similar code if node.left == Literal(true)
        }
    }
}
```
Implementing Rules

Rules are often grouped into *phases*  
   » E.g. simplify Boolean expressions, pushdown selects, choose join algorithms, etc

Each phase runs rules till they no longer apply

```java
plan = originalPlan;
while (true) {
    for (rule in rules) {
        rule.apply(plan);
    }
    if (plan was not changed by any rule) break;
}
```
Simple rules can work together to optimize complex query plans (if designed well):

```sql
SELECT * FROM users WHERE
  (age>=16 && loc==CA) || (age>=16 && loc==NY) || age>=18

  (age>=16) && (loc==CA || loc==NY) || age>=18

  (age>=16 && (loc IN (CA, NY))) || age>=18

  age>=18 || (age>=16 && (loc IN (CA, NY)))
```
Example Extensible Optimizer

For Thursday, you’ll read about Spark SQL’s Catalyst optimizer
  » Written in Scala using its pattern matching features to simplify writing rules
  » >500 contributors worldwide, >1000 types of expressions, and hundreds of rules

We’ll modify Spark SQL in assignment 2
/**
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements. See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License. You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 */

package org.apache.spark.sql.catalyst.optimizer

import scala.collection.mutable
/**
 * Defines the default rule batches in the Optimizer.
 * Implementations of this class should override this method, and [[nonExcludableRules]] if
 * necessary, instead of [[batches]]. The rule batches that eventually run in the Optimizer,
 * i.e., returned by [[batches]], will be (defaultBatches - (excludedRules - nonExcludableRules)).
 */

def defaultBatches: Seq[Batch] = {
  val operatorOptimizationRuleSet = Seq(
    // Operator push down
    PushProjectionThroughUnion,
    ReorderJoin,
    EliminateOuterJoin,
    PushDownPredicates,
    PushDownLeftSemiAntiJoin,
    PushLeftSemiLeftAntiThroughJoin,
    LimitPushDown,
    LimitPushDownThroughWindow,
    ColumnPruning,
    GenerateOptimization,
    // Operator combine
    CollapseRepartition,
    CollapseProject,
    OptimizeWindowFunctions,
    CollapseWindow,
    CombineFilters,
    EliminateLimits,
    CombineUnions,
    // Constant folding and strength reduction
    OptimizeRepartition,
    TransposeWindow,
    NullPropagation,
    NullDownPropagation,
    ConstantPropagation,
    FoldablePropagation,
    OptimizeIn,
    ConstantFolding,
    EliminateAggregateFilter,
    ReorderAssociativeOperator,
    LikeSimplification,
    NotPropagation,
    BooleanSimplification,
    SimplifyConditionals,
    PushFoldableIntoBranches,
    RemoveDispensableExpressions,
    SimplifyBinaryComparison,
    ReplaceNullWithFalseInPredicate,
    SimplifyConditionalsInPredicate,
    PruneFilters,
    SimplifyCasts,
    SimplifyCaseConversionExpressions,
    RewriteCorrelatedScalarSubquery,
    RewriteLateralSubquery,
  )
}
Common Rule-Based Optimizations

Simplifying expressions in select, project, etc
  » Boolean algebra, numeric expressions, string expressions, etc
  » Many redundancies because queries are optimized for readability or produced by code

Simplifying relational operator graphs
  » Select, project, join, etc

These relational optimizations have the most impact
Common Rule-Based Optimizations

Selecting access paths and operator implementations in simple cases

» Index column predicate ⇒ use index

» Small table ⇒ use hash join against it

» Aggregation on field with few values ⇒ use in-memory hash table

Rules also often used to do type checking and analysis (easy to write recursively)
Common Relational Rules

Push selects as far down the plan as possible

Recall:

\[ \sigma_p(R \bowtie S) = \sigma_p(R) \bowtie S \]
if \( p \) only references \( R \)

\[ \sigma_q(R \bowtie S) = R \bowtie \sigma_q(S) \]
if \( q \) only references \( S \)

\[ \sigma_{p \land q}(R \bowtie S) = \sigma_p(R) \bowtie \sigma_q(S) \]
if \( p \) on \( R \), \( q \) on \( S \)

Idea: reduce \# of records early to minimize work in later ops; enable index access paths
Common Relational Rules

Push projects as far down as possible

Recall:

$$\Pi_x(\sigma_p(R)) = \Pi_x(\sigma_p(\Pi_{x \cup z}(R)))$$  \hspace{1cm} z = \text{the fields in } p$$

$$\Pi_{x \cup y}(R \bowtie S) = \Pi_{x \cup y}((\Pi_{x \cup z}(R)) \bowtie (\Pi_{y \cup z}(S)))$$

$$x = \text{fields in } R, \ y = \text{in } S, \ z = \text{in both}$$

Idea: don’t process fields you’ll just throw away
Project Rules Can Backfire!

Example: R has fields A, B, C, D, E
p: A=3 ∧ B="cat"
x: {E}

\[ \Pi_x(\sigma_p(R)) \quad vs \quad \Pi_x(\sigma_p(\Pi_{\{A,B,E\}}(R))) \]
What if R has Indexes?

A = 3

B = “cat”

Intersect buckets to get pointers to matching tuples

In this case, should do $\sigma_p(R)$ first!
Bottom Line

Many valid transformations will not always improve performance

Need more info to make good decisions

» **Data statistics:** properties about our input or intermediate data to be used in planning

» **Cost models:** how much time will an operator take given certain input data statistics?
Outline

What can we optimize?

Rule-based optimization

Data statistics

Cost models

Cost-based plan selection
What Are Data Statistics?

Information about the tuples in a relation that can be used to estimate size & cost

» Example: # of tuples, average size of tuples, # distinct values for each attribute, % of null values for each attribute

Typically maintained by the storage engine as tuples are added & removed in a relation

» File formats like Parquet can also have them
Some Statistics We’ll Use

For a relation $R$,

$$T(R) = \# \text{ of tuples in } R$$

$$S(R) = \text{average size of } R\text{’s tuples in bytes}$$

$$B(R) = \# \text{ of blocks to hold all of } R\text{’s tuples}$$

$$V(R, A) = \# \text{ distinct values of attribute } A \text{ in } R$$
Example

\begin{tabular}{|c|c|c|c|}
\hline
\textbf{A} & \textbf{B} & \textbf{C} & \textbf{D} \\
\hline
\text{cat} & 1 & 10 & a \\
\text{cat} & 1 & 20 & b \\
\text{dog} & 1 & 30 & a \\
\text{dog} & 1 & 40 & c \\
\text{bat} & 1 & 50 & d \\
\hline
\end{tabular}

A: 20 byte string
B: 4 byte integer
C: 8 byte date
D: 5 byte string
Example

\[ \begin{array}{c|c|c|c|c}
R: & A & B & C & D \\
\hline
\text{cat} & 1 & 10 & a & \text{A: 20 byte string} \\
\text{cat} & 1 & 20 & b & \text{B: 4 byte integer} \\
\text{dog} & 1 & 30 & a & \text{C: 8 byte date} \\
\text{dog} & 1 & 40 & c & \text{D: 5 byte string} \\
\text{bat} & 1 & 50 & d & \\
\end{array} \]

\[T(R) = 5\]  \[S(R) = 37\]
\[V(R, A) = 3\]  \[V(R, C) = 5\]
\[V(R, B) = 1\]  \[V(R, D) = 4\]
Challenge: Intermediate Tables

Keeping stats for tables on disk is easy, but what about intermediate tables that appear during a query plan?

Examples:

\[ \sigma_p(R) \quad \text{We already have } T(R), S(R), V(R, a), \text{ etc, but how to get these for tuples that pass } p? \]

\[ R \bowtie S \quad \text{How many and what types of tuple pass the join condition?} \]

Should we do \((R \bowtie S) \bowtie T\) or \(R \bowtie (S \bowtie T)\) or \((R \bowtie T) \bowtie S\)?
Stat Estimation Methods

Algorithms to estimate subplan stats

An ideal algorithm would have:

1) Accurate estimates of stats
2) Low cost
3) Consistent estimates (e.g. different plans for a subtree give same estimated stats)

Can’t always get all this!
Size Estimates for $W = R_1 \times R_2$

$S(W) =$

$T(W) =$
Size Estimates for $W = R_1 \times R_2$

$S(W) = S(R_1) + S(R_2)$

$T(W) = T(R_1) \times T(R_2)$
Size Estimate for $W = \sigma_{A=a}(R)$

$S(W) =$

$T(W) =$
Size Estimate for $W = \sigma_{A=a}(R)$

$S(W) = S(R)$  \hspace{1cm} \text{Not true if some variable-length fields are correlated with value of A}$

$T(W) =$
Example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>1</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>30</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>40</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>bat</td>
<td>1</td>
<td>50</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

$W = \sigma_{Z=\text{val}}(R)$

$T(W) =$

$V(R,A) = 3$

$V(R,B) = 1$

$V(R,C) = 5$

$V(R,D) = 4$
Example

\[
W = \sigma_{Z=\text{val}}(R) \quad T(W) =
\]

\[
R
\]

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>1</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>30</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>40</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>bat</td>
<td>1</td>
<td>50</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

\[
V(R,A)=3 \quad V(R,B)=1 \quad V(R,C)=5 \quad V(R,D)=4
\]

what is probability this tuple will be in answer?
**Example**

\[
\begin{array}{cccc}
  & A & B & C & D \\
\text{cat} & 1 & 10 & a & \text{V(R,A)}=3 \\
\text{cat} & 1 & 20 & b & \text{V(R,B)}=1 \\
\text{dog} & 1 & 30 & a & \text{V(R,C)}=5 \\
\text{dog} & 1 & 40 & c & \text{V(R,D)}=4 \\
\text{bat} & 1 & 50 & d & \\
\end{array}
\]

\[
W = \sigma_{Z=\text{val}}(R) \quad T(W) = \frac{T(R)}{V(R,Z)}
\]
Assumption:

Values in select expression $Z=\text{val}$ are **uniformly distributed** over all $V(R, Z)$ values.
Alternate Assumption:

Values in select expression $Z=\text{val}$ are uniformly distributed over a domain with $\text{DOM}(R, Z)$ values
Example

$$W = \sigma_{Z=\text{val}}(R)$$

$T(W) =$

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>1</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>30</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>40</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>bat</td>
<td>1</td>
<td>50</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

Alternate assumption

$V(R,A)=3,\ DOM(R,A)=10$

$V(R,B)=1,\ DOM(R,B)=10$

$V(R,C)=5,\ DOM(R,C)=10$

$V(R,D)=4,\ DOM(R,D)=10$
Example

\[ W = \sigma_{Z=\text{val}}(R) \]

\[ T(W) = \]

Alternate assumption

\[ V(R,A) = 3, \ DOM(R,A) = 10 \]
\[ V(R,B) = 1, \ DOM(R,B) = 10 \]
\[ V(R,C) = 5, \ DOM(R,C) = 10 \]
\[ V(R,D) = 4, \ DOM(R,D) = 10 \]

what is probability this tuple will be in answer?
Example

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>1</td>
<td>10</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
<td>20</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>30</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>40</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>bat</td>
<td>1</td>
<td>50</td>
<td>d</td>
<td></td>
</tr>
</tbody>
</table>

Alternate assumption

$V(R,A) = 3$, $DOM(R,A) = 10$
$V(R,B) = 1$, $DOM(R,B) = 10$
$V(R,C) = 5$, $DOM(R,C) = 10$
$V(R,D) = 4$, $DOM(R,D) = 10$

$W = \sigma_{Z=val}(R)$

$T(W) = \frac{T(R)}{DOM(R,Z)}$
Selection Cardinality

$SC(R, A) = \text{average } \# \text{ records that satisfy equality condition on } R.A$

$SC(R,A) = \left\{ \begin{array}{c}
\frac{T(R)}{V(R,A)} \\
\frac{T(R)}{\text{DOM}(R,A)}
\end{array} \right.$
What About $W = \sigma_z \geq \text{val}(R)$?

$T(W) =$ ?
What About $W = \sigma_{z \geq \text{val}(R)}$?

$T(W) = ?$

Solution 1: $T(W) = T(R) / 2$
What About $W = \sigma_{z \geq \text{val}(R)}$?

$T(W) = \ ?$

Solution 1: $T(W) = T(R) / 2$

Solution 2: $T(W) = T(R) / 3$
Solution 3: Estimate Fraction of Values in Range

Example:

<table>
<thead>
<tr>
<th>R</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min=1</td>
<td>V(R,Z)=10</td>
</tr>
<tr>
<td>Max=20</td>
<td>W = σ_{Z \geq 15}(R)</td>
</tr>
</tbody>
</table>

\[
f = \frac{20-15+1}{20-1+1} = \frac{6}{20}
\]

(fraction of range)

\[T(W) = f \times T(R)\]
Solution 3: Estimate Fraction of Values in Range

Equivalently, if we know values in column:

\[ f = \text{fraction of distinct values } \geq \text{val} \]

\[ T(W) = f \times T(R) \]
What About More Complex Expressions?

E.g. estimate selectivity for

```
SELECT * FROM R
    WHERE user_defined_func(a) > 10
```
else if (is_funcclause(clause))
{
    /*
     * This is not an operator, so we guess at the selectivity. THIS IS A
     * HACK TO GET V4 OUT THE DOOR.  FUNCS SHOULD BE ABLE TO HAVE
     * SELECTIVITIES THEMSELVES.        -- JMH 7/9/92
     */
    s1 = (Selectivity) 0.3333333;
}
function selectivity(PlannerInfo *root,
    Oid funcid,
    List *args,
    Oid inputcollid,
    bool is_join,
    int varRelid,
    JoinType jointype,
    SpecialJoinInfo *sjinfo)
{
    RegProcedure prosupport = get_func_support(funcid);
    SupportRequestSelectivity req;
    SupportRequestSelectivity *sresult;

    /* If no support function is provided, use our historical default
    * estimate, 0.333333. This seems a pretty unprincipled choice, but
    * Postgres has been using that estimate for function calls since 1992.
    * The hoariness of this behavior suggests that we should not be in too
    * much hurry to use another value.
    */
    if (!prosupport)
        return (Selectivity) 0.333333;

    req.type = T_SupportRequestSelectivity;
    req.root = root;
    req.funcid = funcid;
    req.args = args;
    req.inputcollid = inputcollid;
    req.is_join = is_join;
    req.varRelid = varRelid;
    req.jointype = jointype;
    req.sjinfo = sjinfo;
    req.selectivity = -1;    /* to catch failure to set the value */

    sresult = (SupportRequestSelectivity *)
        DatumGetPointer[OidFunctionCall1(prosupport,
            PointerGetDatum(&req))];

    /* If support function fails, use default */
    if (sresult != &req)
        return (Selectivity) 0.333333;

    if (req.selectivity < 0.0 || req.selectivity > 1.0)
        elog(ERROR, "invalid function selectivity: %f", req.selectivity);

    return (Selectivity) req.selectivity;
Size Estimate for $W = R_1 \bowtie R_2$

Let $X = \text{attributes of } R_1$

\[ Y = \text{attributes of } R_2 \]

Case 1: $X \cap Y = \emptyset$:

Same as $R_1 \times R_2$
Case 2: $W = R_1 \bowtie R_2$, $X \cap Y = A$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th></th>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td></td>
<td></td>
<td></td>
<td>$R_2$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case 2: $W = R_1 \bowtie R_2, X \cap Y = A$

$R_1$ | A | B | C | $R_2$ | A | D

Assumption (“containment of value sets”):

$V(R_1, A) \leq V(R_2, A) \implies$ Every A value in $R_1$ is in $R_2$

$V(R_2, A) \leq V(R_1, A) \implies$ Every A value in $R_2$ is in $R_1$
Computing $T(W)$ when $V(R_1, A) \leq V(R_2, A)$

<table>
<thead>
<tr>
<th>$R_1$</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_2</td>
<td>A</td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

Take 1 tuple

\[ 1 \text{ tuple matches with } T(R_2) \text{ tuples...} \]

\[ \frac{V(R_2, A)}{V(R_2, A)} \]

so \[ T(W) = \frac{T(R_1) \times T(R_2)}{V(R_2, A)} \]
\[ V(R_1, A) \leq V(R_2, A) \Rightarrow T(W) = \frac{T(R_1) \times T(R_2)}{V(R_2, A)} \]

\[ V(R_2, A) \leq V(R_1, A) \Rightarrow T(W) = \frac{T(R_1) \times T(R_2)}{V(R_1, A)} \]
In General for $W = R_1 \Join R_2$

\[
T(W) = \frac{T(R_1) \times T(R_2)}{\max(V(R_1, A), V(R_2, A))}
\]

Where $A$ is the common attribute set
Case 2 with Alternate Assumption

Values uniformly distributed over domain

\[
\begin{array}{ccc}
R_1 & A & B & C \\
\hline & & & \\
\end{array}
\quad
\begin{array}{ccc}
R_2 & A & D \\
\hline & & \\
\end{array}
\]

This tuple matches \( T(R_2) / \text{DOM}(R_2, A) \), so

\[
T(W) = \frac{T(R_1) T(R_2)}{\text{DOM}(R_2, A)} = \frac{T(R_1) T(R_2)}{\text{DOM}(R_1, A)}
\]

Assume these are the same
Tuple Size after Join

In all cases:

\[ S(W) = S(R_1) + S(R_2) - S(A) \]

size of attribute A
Using Similar Ideas, Can Estimate Sizes of:

\[ \Pi_{A,B}(R) \]

\[ \sigma_{A=a \land B=b}(R) \]

R \bowtie S  with common attributes A, B, C

Set union, intersection, difference, …
For Complex Expressions, Need Intermediate $T$, $S$, $V$ Results

E.g. $W = \sigma_{A=a}(R_1) \bowtie R_2$

Treat as relation $U$

$T(U) = T(R_1) / V(R_1, A)$ \hspace{1cm} $S(U) = S(R_1)$

Also need $V(U, *)$ !!
To Estimate $V$

E.g., $U = \sigma_{A=a}(R_1)$

Say $R_1$ has attributes $A, B, C, D$

$V(U, A) =$

$V(U, B) =$

$V(U, C) =$

$V(U, D) =$
Example

R_1

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>1</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>30</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>40</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>bat</td>
<td>1</td>
<td>50</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

V(R_1, A) = 3
V(R_1, B) = 1
V(R_1, C) = 5
V(R_1, D) = 3

U = \sigma_{A=a}(R_1)
**Example**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>1</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>cat</td>
<td>1</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>30</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>40</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>bat</td>
<td>1</td>
<td>50</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

V(R₁, A) = 3
V(R₁, B) = 1
V(R₁, C) = 5
V(R₁, D) = 3

U = σₐ(A)(R₁)

V(U, A) = 1  V(U, B) = 1  V(U, C) = \frac{T(R₁)}{V(R₁, A)}

V(U, D) = somewhere in between…
Possible Guess in $U = \sigma_{A \geq a}(R)$

$V(U, A) = V(R, A) / 2$

$V(U, B) = V(R, B)$
For Joins: $U = R_1(A, B) \bowtie R_2(A, C)$

We’ll use the following estimates:

\[ V(U, A) = \min(V(R_1, A), V(R_2, A)) \]

\[ V(U, B) = V(R_1, B) \]

\[ V(U, C) = V(R_2, C) \]

Called “preservation of value sets”
Example:

$$Z = R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,D)$$

<table>
<thead>
<tr>
<th>$R_1$</th>
<th>$T(R_1) = 1000$</th>
<th>$V(R_1,A) = 50$</th>
<th>$V(R_1,B) = 100$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_2$</td>
<td>$T(R_2) = 2000$</td>
<td>$V(R_2,B) = 200$</td>
<td>$V(R_2,C) = 300$</td>
</tr>
<tr>
<td>$R_3$</td>
<td>$T(R_3) = 3000$</td>
<td>$V(R_3,C) = 90$</td>
<td>$V(R_3,D) = 500$</td>
</tr>
</tbody>
</table>
Partial Result: $U = R_1 \boxtimes R_2$

$T(U) = \frac{1000 \times 2000}{200}$

$V(U,A) = 50$

$V(U,B) = 100$

$V(U,C) = 300$
End Result: $Z = U \boxtimes R_3$

$T(Z) = \frac{1000 \times 2000 \times 3000}{200 \times 300}$

$V(Z,A) = 50$
$V(Z,B) = 100$
$V(Z,C) = 90$
$V(Z,D) = 500$
Another Statistic: Histograms

number of tuples in R with A value in a given range

\[ \sigma_{A \geq a}(R) = ? \]
\[ \sigma_{A = a}(R) = ? \]

Requires some care to set bucket boundaries
Outline

What can we optimize?

Rule-based optimization

Data statistics

Cost models

Cost-based plan selection