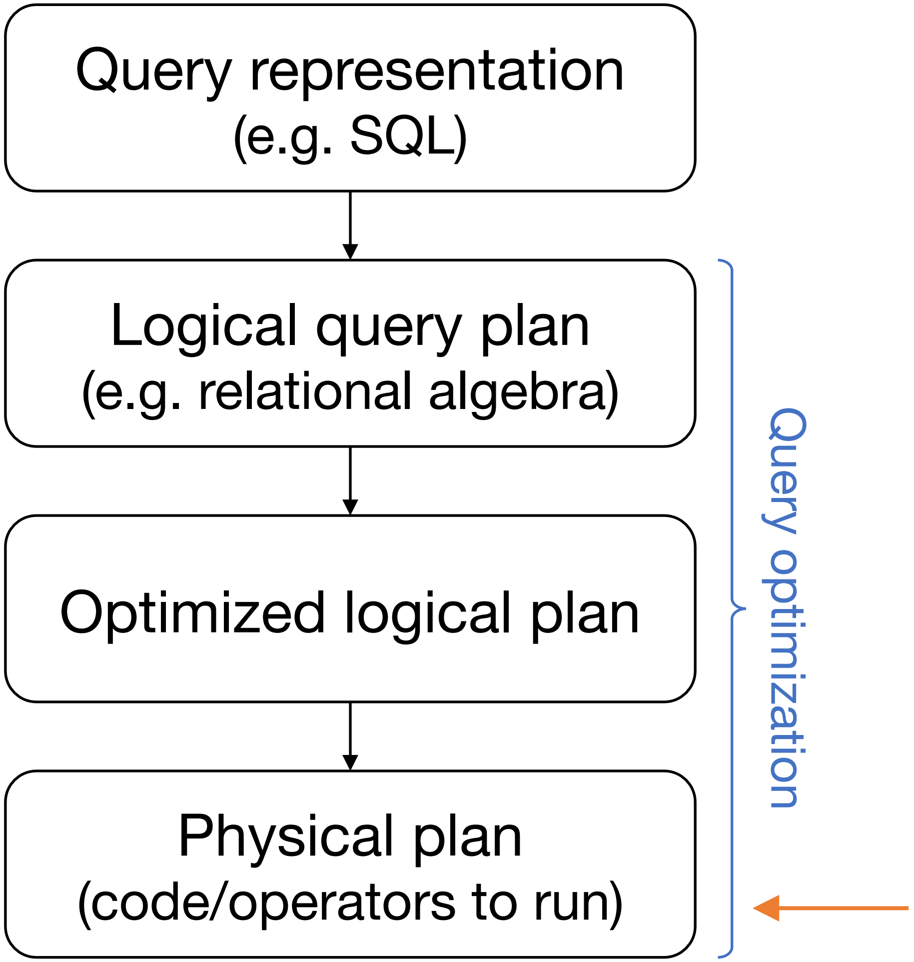
Query Execution 2 and Query Optimization

Instructor: Matei Zaharia

cs245.stanford.edu

Query Execution Overview



Execution Methods: Once We Have a Plan, How to Run it?

Several options that trade between complexity, performance and startup time

Method 1: Interpretation

```
interface Operator {
  Tuple next();
}

class TableScan: Operator

class Select: Operator

class Project: Operator

...
```

```
interface Expression {
  Value compute(Tuple in);
}

class Attribute: Expression

class Times: Expression

class Equals: Expression

...
```

Running Our Query with Interpretation

```
ops = Project(
        expr = Times(Attr(“quantity”), Attr(“price”)),
        parent = Select(
          expr = Equals(Attr(“productId”), Literal(75)),
          parent = TableScan(“orders”)
        )
      );

while(true) {
  Tuple t = ops.next();
  if (t != null) {
    out.write(t);
  } else {
    break;
  }
}
```

recursively calls Operator.next() and Expression.compute()

Method 2: Vectorization

Interpreting query plans one record at a time is simple, but it's too slow

» Lots of virtual function calls and branches for each record (recall Jeff Dean's numbers)

Keep recursive interpretation, but make Operators and Expressions run on **batches**

Implementing Vectorization

```
class TupleBatch {
  // Efficient storage, e.g.
  // schema + column arrays
}

interface Operator {
  TupleBatch next();
}

class Select: Operator {
  Operator parent;
  Expression condition;
}

...
```

```
class ValueBatch {
  // Efficient storage
}

interface Expression {
  ValueBatch compute(
    TupleBatch in);
}

class Times: Expression {
  Expression left, right;
}

...
```

Typical Implementation

Values stored in columnar arrays (e.g. int[]) with a separate bit array to mark nulls

Tuple batches fit in L1 or L2 cache

Operators use SIMD instructions to update both values and null fields without branching

Pros & Cons of Vectorization

+ Faster than record-at-a-time if the query processes many records

+ Relatively simple to implement

– Lots of nulls in batches if query is selective

– Data travels between CPU & cache a lot

Method 3: Compilation

Turn the query into executable code

Compilation Example

$\Pi_{\text{quanity*price}} (\sigma_{\text{productId=75}} (\text{orders}))$

generated class with the right field types for orders table

```
class MyQuery {
  void run() {
    Iterator<OrdersTuple> in = openTable(“orders”);
    for(OrdersTuple t: in) {
      if (t.productId == 75) {
        out.write(Tuple(t.quantity * t.price));
      }
    }
  }
}
```

Can also theoretically generate vectorized code

Pros & Cons of Compilation

+ Potential to get fastest possible execution

+ Leverage existing work in compilers

– Complex to implement

– Compilation takes time

– Generated code may not match hand-written

What's Used Today?

Depends on context & other bottlenecks

Transactional databases (e.g. MySQL): mostly record-at-a-time interpretation

Analytical systems (Vertica, Spark SQL): vectorization, sometimes compilation

ML libs (TensorFlow): mostly vectorization (the records *are* vectors!), some compilation

Query Optimization

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?

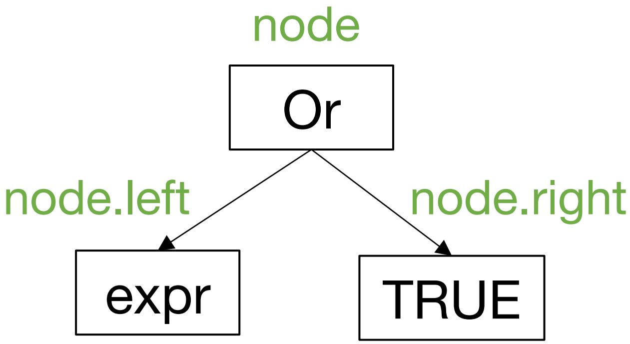
Procedure to replace part of the query plan based on a pattern seen in the plan

Example: When I see expr OR TRUE for an expression expr, replace this with TRUE

Implementing Rules

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

```
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

```
plan = originalPlan;
while (true) {
  for (rule in rules) {
    rule.apply(plan);
  }
  if (plan was not changed by any rule) break;
}
```

Result

Simple rules can work together to optimize complex query plans (if designed well):

```
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

Search or jump to... / Pull requests Issues Marketplace Explore

apache / spark

Watch 2.1k | Unstar 28.7k | Fork 23.3k

Code | Pull requests 232 | Actions | Projects | Security | Insights

master ▾ spark / sql / catalyst / src / main / scala / org / apache / spark / sql / catalyst / optimizer / Optimizer.scala

Go to file | ···

yaooqinn [SPARK-34083][SQL] Using TPCDS original definitions for char/varchar ... ✓ Latest commit 5718d64 7 days ago History

113 contributors +69

1908 lines (1735 sloc) | 80.4 KB

Raw | Blame

```
/*
 * 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
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

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

import scala.collection.mutable

import org.apache.spark.sql.AnalysisException
import org.apache.spark.sql.catalyst.analysis._
import org.apache.spark.sql.catalyst.catalog.{InMemoryCatalog, SessionCatalog}
```

```scala
  /**
   * 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,
        ColumnPruning,
        // Operator combine
        CollapseRepartition,
        CollapseProject,
        OptimizeWindowFunctions,
        CollapseWindow,
        CombineFilters,
        EliminateLimits,
        CombineUnions,
        // Constant folding and strength reduction
        OptimizeRepartition,
        TransposeWindow,
        NullPropagation,
        ConstantPropagation,
        FoldablePropagation,
        OptimizeIn,
        ConstantFolding,
        EliminateAggregateFilter,
        ReorderAssociativeOperator,
        LikeSimplification,
        BooleanSimplification,
        SimplifyConditionals,
        PushFoldableIntoBranches,
        RemoveDispensableExpressions,
        SimplifyBinaryComparison,
        ReplaceNullWithFalseInPredicate,
        SimplifyConditionalsInPredicate,
        PruneFilters,
        SimplifyCasts,
```

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 ← Also very high impact

- 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

$\sigma_{p \wedge q}(R \bowtie S) = \sigma_p(R) \bowtie \sigma_q(S)$ 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:

$\Pi_x(\sigma_p(R)) = \Pi_x(\sigma_p(\Pi_{x \cup z}(R)))$ z = 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 = fields in R, y = in S, z = in both

Idea: don't process fields you'll just throw away

Project Rules Can Backfire!

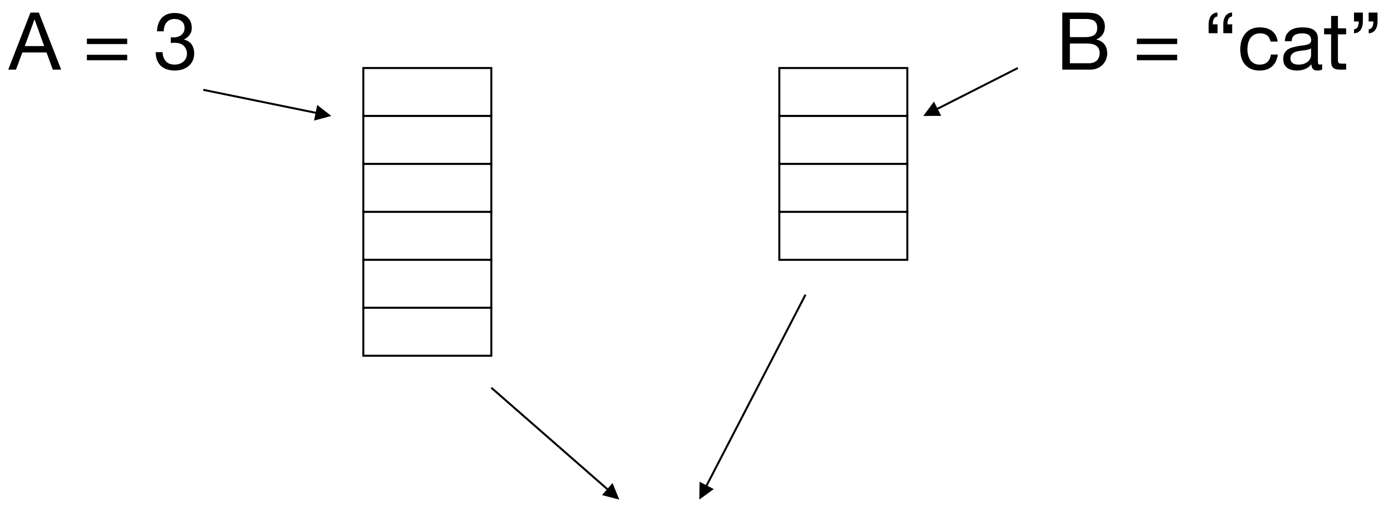
Example: R has fields A, B, C, D, E

p: $A=3 \land B=\text{"cat"}$

x: {E}

$$\Pi_x(\sigma_p(R)) \quad \text{vs} \quad \Pi_x(\sigma_p(\Pi_{\{A,B,E\}}(R)))$$

What if R has Indexes?



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) = # of tuples in R

S(R) = average size of R's tuples in bytes

B(R) = # of blocks to hold all of R's tuples

V(R, A) = # distinct values of attribute A in R

Example

R:

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

A: 20 byte string

B: 4 byte integer

C: 8 byte date

D: 5 byte string

Example

R:

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

A: 20 byte string

B: 4 byte integer

C: 8 byte date

D: 5 byte string

T(R) = 5

V(R, A) = 3

V(R, B) = 1

S(R) = 37

V(R, C) = 5

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:

$\sigma_p(R)$ ⟵ We already have T(R), S(R), V(R, a), etc, but how to get these for tuples that pass p?

$R \bowtie S$ ⟵ 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 $W = \sigma_{A=a}(R)$

$S(W) = S(R)$ ← Not true if some variable-length fields are correlated with value of A

$T(W) =$

Example

R

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

$V(R,A)=3$

$V(R,B)=1$

$V(R,C)=5$

$V(R,D)=4$

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

Example

R

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

V(R,A)=3

V(R,B)=1

V(R,C)=5

V(R,D)=4

what is probability this tuple will be in answer?

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

Example

R

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

$V(R,A)=3$

$V(R,B)=1$

$V(R,C)=5$

$V(R,D)=4$

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

$$T(W) = \frac{T(R)}{V(R,Z)}$$

Assumption:

Values in select expression Z=val are **uniformly distributed** over all V(R, Z) values

Alternate Assumption:

Values in select expression Z=val are **uniformly distributed** over a domain with DOM(R, Z) values

Example

R

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

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) =

Example

R

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

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?

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

Example

R

A	B	C	D
cat	1	10	a
cat	1	20	b
dog	1	30	a
dog	1	40	c
bat	1	50	d

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

$$W = \sigma_{Z=val}(R) \qquad T(W) = \frac{T(R)}{DOM(R,Z)}$$

Selection Cardinality

SC(R, A) = average # records that satisfy equality condition on R.A

$$SC(R,A) = \begin{cases} \dfrac{T(R)}{V(R,A)} \\ \\ \dfrac{T(R)}{DOM(R,A)} \end{cases}$$

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

$T(W) = ?$

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

$T(W) = ?$

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

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

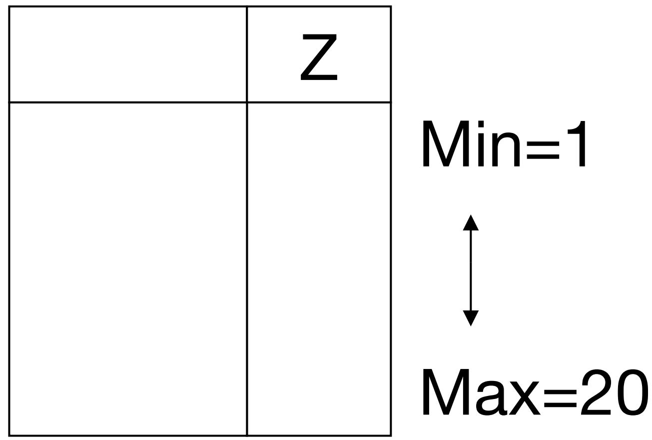
T(W) = ?

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

Solution 2: T(W) = T(R) / 3

Solution 3: Estimate Fraction of Values in Range

Example: R



$V(R,Z)=10$

$W = \sigma_{Z \geq 15}(R)$

$$f = \frac{20-15+1}{20-1+1} = \frac{6}{20}$$ (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 = fraction of distinct values $\geq$ 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
```

postgres / postgres

Watch 267 | Star 2,547 | Fork 820

Code | Pull requests 0 | Projects 0 | Pulse | Graphs

Tree: 4cbe3abb31 postgres / src / backend / optimizer / path / clausesel.c

Find file | Copy path

bmomjian pgindent run for 9.4 0a78320 on May 6, 2014

5 contributors

785 lines (733 sloc) 21.6 KB

Raw | Blame | History

```
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;
}
```

Size Estimate for $W = R_1 \bowtie R_2$

Let X = attributes of R_1

Y = attributes of R_2

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

Same as $R_1 \times R_2$

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

R_1

A	B	C

R_2

A	D

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

R_1

A	B	C

R_2

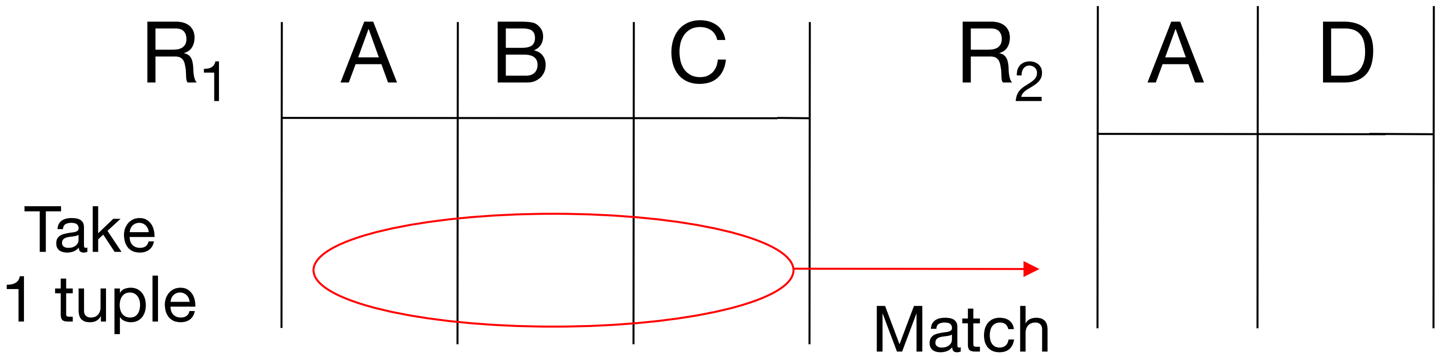
A	D

Assumption ("containment of value sets"):

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

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

Computing T(W) when $V(R_1, A) \le V(R_2, A)$



1 tuple matches with $\frac{T(R_2)}{V(R_2, A)}$ tuples...

so $T(W) = \frac{T(R_1) \times T(R_2)}{V(R_2, A)}$

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

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

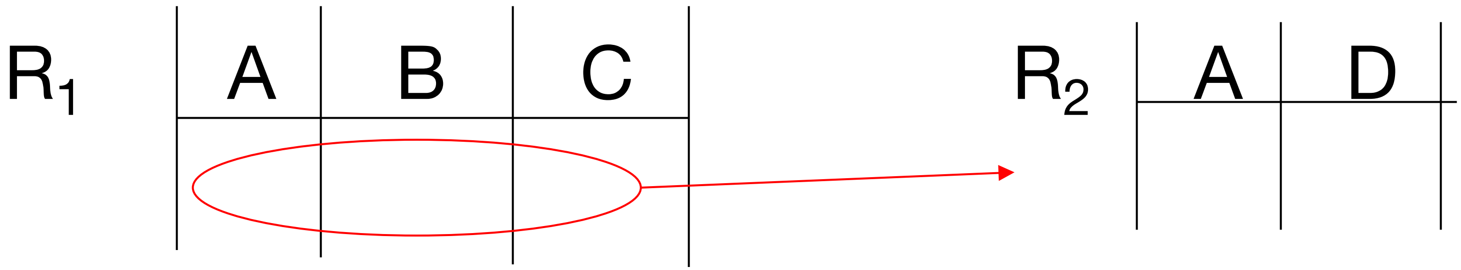
In General for $W = R_1 \bowtie R_2$

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

Where A is the common attribute set

Case 2 with Alternate Assumption

Values uniformly distributed over domain



This tuple matches $T(R_2) / DOM(R_2, A)$, so

$$T(W) = \frac{T(R_1)\,T(R_2)}{DOM(R_2, A)} = \frac{T(R_1)\,T(R_2)}{DOM(R_1, A)}$$

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:

$\Pi_{A,B}(R)$

$\sigma_{A=a \wedge B=b}(R)$

$R \bowtie S$ 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

$T(U) = T(R_1) / V(R_1, A)$ $S(U) = S(R_1)$

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

A	B	C	D
cat	1	10	10
cat	1	20	20
dog	1	30	10
dog	1	40	30
bat	1	50	10

$V(R_1, A)=3$

$V(R_1, B)=1$

$V(R_1, C)=5$

$V(R_1, D)=3$

$U = \sigma_{A=a}(R_1)$

Example

R_1

A	B	C	D
cat	1	10	10
cat	1	20	20
dog	1	30	10
dog	1	40	30
bat	1	50	10

$V(R_1, A)=3$

$V(R_1, B)=1$

$V(R_1, C)=5$

$V(R_1, D)=3$

$U = \sigma_{A=a}(R_1)$

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

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:

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

R_1	$T(R_1) = 1000$ $V(R_1,A)=50$ $V(R_1,B)=100$
R_2	$T(R_2) = 2000$ $V(R_2,B)=200$ $V(R_2,C)=300$
R_3	$T(R_3) = 3000$ $V(R_3,C)=90$ $V(R_3,D)=500$

Partial Result: $U = R_1 \bowtie R_2$

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

$V(U,A) = 50$

$V(U,B) = 100$

$V(U,C) = 300$

End Result: $Z = U \bowtie R_3$

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

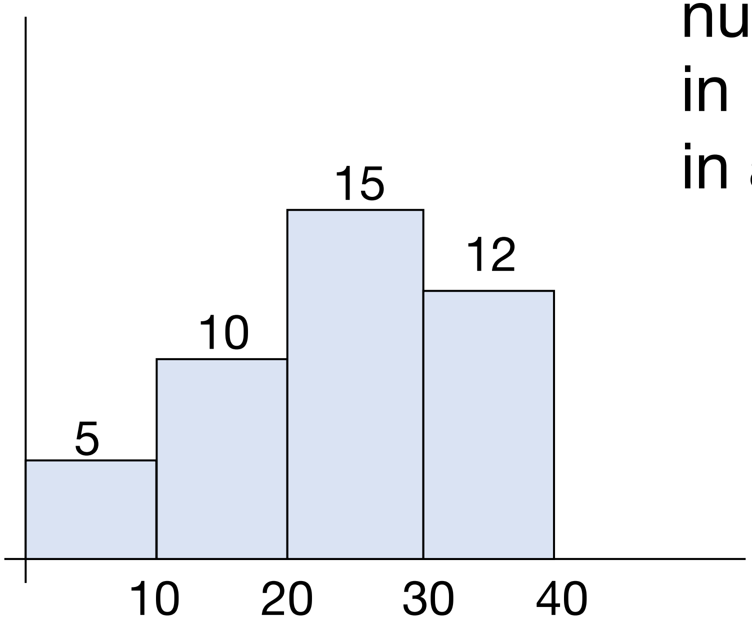
$V(Z,A) = 50$

$V(Z,B) = 100$

$V(Z,C) = 90$

$V(Z,D) = 500$

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