Query Execution 2 and Query Optimization

Instructor: Matei Zaharia

cs245.stanford.edu
Query Execution Overview

1. Query representation (e.g. SQL)
2. Logical query plan (e.g. relational algebra)
3. Optimized logical plan
4. Physical plan (code/operators to run)
Example SQL Query

SELECT title
FROM StarsIn
WHERE starName IN (
    SELECT name
    FROM MovieStar
    WHERE birthdate LIKE '%%1960'
);

(Find the movies with stars born in 1960)
SELECT <SelList> FROM <FromList> WHERE <Condition>

<title> title </title>
星星In <Attribute> ( <Query> )

starName <SFW>

SELECT <SelList> FROM <FromList> WHERE <Condition>

<Attribute> LIKE <Pattern>

name MovieStar birthDate ‘%1960’
Logical Query Plan

\[ \Pi_{\text{title}} \]

\[ \sigma_{\text{starName}=\text{name}} \]

\[ \times \]

\[ \Pi_{\text{name}} \]

\[ \sigma_{\text{birthdate} \ LIKE \ '%1960'} \]

\[ \times \]

\[ \text{MovieStar} \]
Improved Logical Query Plan

\[\Pi_{title}(\sigma_{\text{birthdate} \text{ LIKE } '1960'}(\Pi_{name}(\text{StarsIn}(\text{MovieStar}))))\]
Estimate Result Sizes

Need expected size

StarsIn

MovieStar
One Physical Plan

Hash join

Seq scan
StarsIn

Index scan
MovieStar

Parameters: join order, memory size, project attributes, ...

Parameters: select condition, ...
Another Physical Plan

Hash join

Parameters: join order, memory size, project attributes, ...

Index scan

Parameters: select condition, ...

StarsIn

Seq scan

MovieStar

H
Another Physical Plan

Sort-merge join

Seq scan

StarsIn

Seq scan

MovieStar
Estimating Plan Costs

Logical plan

Physical plan candidates

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

Several options that trade between complexity, performance and startup time
Example: Simple Query

SELECT quantity * price
    FROM orders
    WHERE productId = 75

\[\Pi_{quantity\times price} (\sigma_{productId=75} (\text{orders}))\]
Method 1: Interpretation

interface Operator {
    Tuple next();
}

class TableScan: Operator {
    String tableName;
}

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

class Project: Operator {
    Operator parent;
    Expression[] exprs;
}

interface Expression {
    Value compute(Tuple in);
}

class Attribute: Expression {
    String name;
}

class Times: Expression {
    Expression left, right;
}

class Equals: Expression {
    Expression left, right;
}
Example Expression Classes

class Attribute: Expression {
    String name;
    Value compute(Tuple in) {
        return in.getField(name);
    }
}

class Times: Expression {
    Expression left, right;
    Value compute(Tuple in) {
        return left.compute(in) * right.compute(in);
    }
}
Example Operator Classes

class TableScan: Operator {
    String tableName;

    Tuple next() {
        // read & return next record from file
        }
    }

class Project: Operator {
    Operator parent;
    Expression[] exprs;

    Tuple next() {
        tuple = parent.next();
        fields = [expr.compute(tuple) for expr in exprs];
        return new Tuple(fields);
    }
    }
Running Our Query with Interpretation

```java
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;
    }
}
```

Pros & cons of this approach?
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;
}

...
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

\[ \Pi_{\text{quantity} \times \text{price}} (\sigma_{\text{productId} = 75} (\text{orders})) \]

generated class with the right field types for orders table

```java
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 \times Access paths for table 2 \times Algorithms for join 1 \times Algorithms for join 2 \times \ldots

**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)
        }
    }
}

// Example query plan

node
    Or
    expr
    TRUE
    node.left
    node.right
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

```cpp
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):

```
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 also use Spark SQL in assignment 2
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}
import org.apache.spark.sql.catalyst.expressions._
import org.apache.spark.sql.catalyst.expressions.aggregate._

/*
 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.
 */
abstract class Optimizer(sessionCatalog: SessionCatalog) {
  extends RuleExecutor[LogicalPlan] {

    def defaultBatches: Seq[Batch] = {
      val operatorOptimizationRuleSet = Seq(
        // Operator push down
        PushProjectionThroughUnion,
        ReorderJoin,
        EliminateOuterJoin,
        PushPredicateThroughJoin,
        PushDownPredicate,
        PushDownLeftSemiAntiJoin,
        PushLeftSemiLeftAntiThroughJoin,
        LimitPushDown,
        ColumnPruning,
        InferFiltersFromConstraints,
        // Operator combine
        CollapseRepartition,
        CollapseProject,
        CollapseWindow,
        CombineFilters,
        CombineLimits,
        CombineUnions,
        // Constant folding and strength reduction
        TransposeWindow,
        NullPropagation,
        ConstantPropagation,
        FoldablePropagation,
        OptimizeIn,
        ConstantFolding,
        ReorderAssociativeOperator,
        LikeSimplification,
        BooleanSimplification,
        SimplifyConditionals,
        RemoveDispensableExpressions,
        SimplifyBinaryComparison,
        ReplaceNullWithFalseInPredicate,
        PruneFilters,
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 generated 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:

$$\Pi_x(\sigma_p(R)) = \Pi_x(\sigma_p(\Pi_{x \cup z}(R)))$$

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

**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 possible transformations aren’t always good for 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

<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>

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

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

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

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

Examples:

\[ \sigma_p(R) \]  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 $W = \sigma_{A=a}(R)$

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

$T(W) =$
**Example**

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

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>cat</td>
<td>1</td>
<td>20</td>
<td>b</td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>30</td>
<td>a</td>
</tr>
<tr>
<td>dog</td>
<td>1</td>
<td>40</td>
<td>c</td>
</tr>
<tr>
<td>bat</td>
<td>1</td>
<td>50</td>
<td>d</td>
</tr>
</tbody>
</table>

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

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

$V(R,A) = 3$

$V(R,B) = 1$

$V(R,C) = 5$

$V(R,D) = 4$

What is the probability this tuple will be in the answer?
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>

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

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

Values in select expression $Z=val$ are uniformly distributed over all $V(R, Z)$ values
Alternate Assumption:

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

Alternate assumption

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

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

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

<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>

Alternate assumption

- $V(R,A)=3$, $\text{DOM}(R,A)=10$
- $V(R,B)=1$, $\text{DOM}(R,B)=10$
- $V(R,C)=5$, $\text{DOM}(R,C)=10$
- $V(R,D)=4$, $\text{DOM}(R,D)=10$

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>

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

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

Alternate assumption:

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

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

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

$V(R,D)=4, \text{ DOM}(R,D)=10$
Selection Cardinality

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

\[
SC(R,A) = \begin{cases} 
T(R) \\
\frac{V(R,A)}{V(R,A)} \\
\frac{T(R)}{\text{DOM}(R,A)} 
\end{cases}
\]
What About $W = \sigma_{z \geq val(R)}$?

$T(W) = ?$
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;
}
Size Estimate for $W = R_1 \Join R_2$

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

$\quad Y = \text{attributes of } R_2$

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

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

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

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

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 $V(R_1, A) \leq V(R_2, A)$

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

Take 1 tuple

Match

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

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

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

<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>

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

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

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:

\[ \Pi_{AB}(R) \]

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

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

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

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

\[ T(U) = \frac{T(R_1)}{V(R_1, A)} \quad S(U) = S(R_1) \]

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

<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>R_1</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)

V(U, A) = 1  
V(U, B) = 1  
V(U, C) = \frac{T(R_1)}{V(R_1, A)}  

V(U, D) = \text{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 = R₁(A,B) ⊙ R₂(A,C)

We’ll use the following estimates:

V(U, A) = min(V(R₁, A), V(R₂, A))

V(U, B) = V(R₁, B)

V(U, C) = V(R₂, 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: $U = R_1 \bowtie R_2$

\[
T(U) = \frac{1000 \times 2000}{200} \quad V(U,A) = 50 \\
V(U,B) = 100 \\
V(U,C) = 300
\]
End Result: \( Z = U \Join R_3 \)

\[
T(Z) = \frac{1000 \times 2000 \times 3000}{200 \times 300} \\
V(Z,A) = 50 \\
V(Z,B) = 100 \\
V(Z,C) = 90 \\
V(Z,D) = 500
\]
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
Cost Models

How do we measure a query plan’s cost?

Many possible metrics:

- Number of disk I/Os
- Number of compute cycles
- Combined time metric
- Memory usage
- Bytes sent on network
- ...

We’ll focus on this
Example: Index vs Table Scan

Our query: $\sigma_p(R)$ for some predicate $p$

$s = p$’s selectivity (fraction tuples passing)

Table scan:
- $R$ has $B(R) = T(R) \times S(R)/b$
- blocks on disk
- Cost: $B(R)$ I/Os

Index search:
- Index lookup for $p$ takes $L$ I/Os
- We then have to read part of $R$;
- $Pr[\text{read block i}] 
  \approx 1 - Pr[\text{no match}]$
- $= 1 - (1-s)^{b/S(R)}$
- Cost: $L + (1-(1-s)^{b/S(R)}) B(R)$
Example: Index vs Table Scan

Our query: $\sigma_p(R)$ for some predicate $p$

$s = p$’s selectivity (fraction tuples passing)

$C_{\text{scan}} = B(R)$

$C_{\text{index}} = L + (1-(1-s)^{b/S(R)}) B(R)$

Index good when $s$ is small, or $S(R)$ is large

Index never “much worse” than table scan…
What If Results Were Clustered?

Unclustered: records that match p are spread out uniformly

Clustered: records that match p are close together in R’s file
What If Results Were Clustered?

Unclustered:
records that match p are spread out uniformly

Clustered:
records that match p are close together in R’s file

We’d need to change our estimate of $C_{\text{index}}$:

$C_{\text{index}} = L + s \cdot B(R)$

Less than $C_{\text{scan}}$ even for bigger $s$
More Detail: Join Operators

Join orders and algorithms are often the choices that affect performance the most.

For a multi-way join $R \bowtie S \bowtie T \bowtie \ldots$, each join is selective and order matters a lot.

» Try to eliminate lots of records early

Even for one join $R \bowtie S$, algorithm matters.
Example

```
SELECT order.date, product.price, customer.name
FROM order, product, customer
WHERE order.product_id = product.product_id
  AND order.cust_id = customer.cust_id
  AND product.type = "car"
  AND customer.country = "US"
```

Plan 1:
- Customer (country=US)
  - Order
  - Product (type=car)

Plan 2:
- Product (type=car)
  - Order
  - Customer (country=US)

When is each plan better?
Common Join Algorithms

Iteration (nested loops) join

Merge join

Join with index

Hash join
Iteration Join

for each $r \in R_1$ do
  for each $s \in R_2$ do
    if $r.C = s.C$ then output $(r, s)$

I/Os: one scan of $R_1$ and $T(R_1)$ scans of $R_2$, so cost $= B(R_1) + T(R_1)B(R_2)$

Improvement: read each block of $R_1$ in RAM first then read $R_2$ for cost $B(R_1) + B(R_1)B(R_2)$
Merge Join

If $R_1$ and $R_2$ not sorted by $C$ then sort them

i, j = 1;
while $i \leq T(R_1)$ && $j \leq T(R_2)$ do
    if $R_1[i].C = R_2[j].C$ then outputTuples
    else if $R_1[i].C > R_2[j].C$ then $j += 1$
    else if $R_1[i].C < R_2[j].C$ then $i += 1$
Merge Join

procedure outputTuples:
    while $R_1[i].C == R_2[j].C$ && $i \leq T(R_1)$ do
        $jj = j$
        while $R_1[i].C = R_2[jj].C$ && $jj \leq T(R_2)$ do
            output ($R_1[i], R_2[jj]$)
            $jj \leftarrow jj + 1$
        i $\leftarrow i + 1$
Example

<table>
<thead>
<tr>
<th>i</th>
<th>$R_1[i].C$</th>
<th>$R_2[j].C$</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>30</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>30</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>50</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>52</td>
<td>7</td>
</tr>
</tbody>
</table>
Cost of Merge Join

If $R_1$ and $R_2$ already sorted by $C$, then

$$\text{cost} = B(R_1) + B(R_2) + \# \text{ blocks in result}$$

Might need to go over some records many times

Sorting an on-disk table is usually a 2-pass process with 2 $B(R)$ I/Os ("external sort")
Outline

What can we optimize?

Rule-based optimization

Data statistics

Cost models

Cost-based plan selection
Complete CBO Process

Generate and compare possible query plans

Generate

Prune

Estimate Cost

Select

Query

Plans

Costs

Pick Min
How to Generate Plans?

Simplest way: recursive search of the options for each planning choice
How to Generate Plans?

Can limit search space: e.g. many DBMSes only consider “left-deep” joins

Often interacts well with conventions for specifying join inputs in asymmetric join algorithms (e.g. assume right argument has index)
How to Generate Plans?

Can prioritize searching through the most impactful decisions first
   » E.g. join order is one of the most impactful
How to Prune Plans?

While computing the cost of a plan, throw it away if it is worse than best so far.

Start with a greedy algorithm to find an “OK” initial plan that will allow lots of pruning.
Memoization and Dynamic Programming

During a search through plans, many subplans will appear repeatedly.

Remember cost estimates and statistics ($T(R)$, $V(R, A)$, etc) for those: “memoization”

Can pick an order of subproblems to make it easy to reuse results (dynamic programming)
Resource Cost of CBO

It’s possible for cost-based optimization itself to take longer than running the query!

Need to design optimizer to not take too long
   » That’s why we have shortcuts in stats, etc

Luckily, a few “big” decisions drive most of the query execution time (e.g. join order)