Data-Intensive Computing Systems

Introduction to Query Processing

Shivnath Babu
Query Processing

Declarative SQL Query $\rightarrow$ Query Plan

NOTE: You will not be tested on how well you know SQL. Understanding the SQL introduced in class will be sufficient (a primer follows). SQL is described in Chapter 6, GMUW.

Focus: Relational System (i.e., data is organized as tables, or relations)
SQL Primer

We will focus on SPJ, or Select-Project-Join Queries

Select  <attribute list>
From    <relation list>
Where   <condition list>

Example Filter Query over R(A,B,C):
Select   B
From     R
Where    R.A = “c” ∧ R.C > 10
We will focus on SPJ, or Select-Project-Join-Queries.

Select  <attribute list>
From    <relation list>
Where   <condition list>

Example Join Query over R(A,B,C) and S(C,D,E):
Select   B, D
From     R, S
Where    R.A = “c” ∧ S.E = 2 ∧ R.C = S.C
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th></th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>10</td>
<td></td>
<td>10</td>
<td>x</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>20</td>
<td></td>
<td>20</td>
<td>y</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>10</td>
<td></td>
<td>30</td>
<td>z</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>2</td>
<td>35</td>
<td></td>
<td>40</td>
<td>x</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>3</td>
<td>45</td>
<td></td>
<td>50</td>
<td>y</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Select B,D
From R,S
Where R.A = “c” ∧
S.E = 2 ∧ R.C=S.C

Answer

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
How do we execute this query?

Select B,D
From R,S
Where R.A = "c" ∨ S.E = 2 ∨ R.C=S.C

- Do Cartesian product
- Select tuples
- Do projection
Select B,D From R,S
Where R.A = “c” ∧ S.E = 2 ∧ R.C=S.C

Bingo!
Got one...
Relational Algebra - can be used to describe plans

Ex: Plan I

\[ \Pi_{B,D} \sigma_{R.A=\text{"c"} \land S.E=2 \land R.C=S.C} R \bowtie S \]
Relational Algebra Primer
(Chapter 5, GMUW)

Select: $\sigma_{R.A="c" \land R.C=10}$

Project: $\Pi_{B,D}$

Cartesian Product: $R \times S$

Natural Join: $R \bowtie S$
Relational Algebra - can be used to describe plans

Ex: Plan I

\[ \Pi_{B,D} \left( \sigma_{R.A = "c" \land S.E = 2 \land R.C = S.C} (R \times S) \right) \]

OR: \[ \Pi_{B,D} \left[ \sigma_{R.A = "c" \land S.E = 2 \land R.C = S.C} (R \times S) \right] \]
Another idea:

Plan II

\[ \Pi_{B,D} \sigma_{R.A = "c"} \sigma_{S.E = 2} \]

Select B,D
From R,S

Where R.A = "c" \land S.E = 2 \land R.C = S.C
### Table R
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>d</td>
<td>2</td>
<td>35</td>
</tr>
<tr>
<td>e</td>
<td>3</td>
<td>45</td>
</tr>
</tbody>
</table>

### Table σ(R)
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

### Table σ(S)
<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>x</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>y</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>z</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>

Select B,D From R,S Where R.A = “c” ∧ S.E = 2 ∧ R.C=S.C
Plan III

Use R.A and S.C Indexes

(1) Use R.A index to select R tuples with R.A = “c”

(2) For each R.C value found, use S.C index to find matching tuples

(3) Eliminate S tuples S.E ≠ 2

(4) Join matching R,S tuples, project B,D attributes, and place in result
The diagram illustrates a data processing flow involving two tables, R and S, and an intermediate process with two intermediate relations, I₁ and I₂. The process begins with the table R:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>d</td>
<td>2</td>
<td>35</td>
</tr>
<tr>
<td>e</td>
<td>3</td>
<td>45</td>
</tr>
<tr>
<td>c</td>
<td>7</td>
<td>15</td>
</tr>
</tbody>
</table>

The first intermediate relation, I₁, filters based on the condition A = "c" and outputs the tuple <c, 2, 10>.

The second intermediate relation, I₂, applies a check and outputs the tuple <2, x>.

The final table, S, shows the processed output:

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>x</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>y</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>z</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>y</td>
<td>3</td>
</tr>
</tbody>
</table>

The next tuple to process is <c, 7, 15>.}

---

**Diagram Explanation**

1. **Table R** contains the following data:
   - a 1 10
   - b 1 20
   - c 2 10
   - d 2 35
   - e 3 45
   - c 7 15

2. **Intermediate Relation I₁** filters R based on A = "c" and outputs the tuple <c, 2, 10>.

3. **Intermediate Relation I₂** takes the output from I₁ and applies a check. It outputs the tuple <2, x>.

4. **Table S** is populated with the processed output from I₂:
   - 10 x 2
   - 20 y 2
   - 30 z 2
   - 40 x 1
   - 50 y 3

5. The next tuple to process is <c, 7, 15>.
Overview of Query Processing

- SQL query
  - parse
    - parse tree
    - Query rewriting
      - logical query plan
        - Physical plan generation
          - physical query plan
            - execute
              - result

Query Optimization

Query Execution
Example Query

Select B,D
From R,S
Where R.A = "c" ∧ R.C=S.C
Example: Parse Tree

```
SELECT <SelList> FROM <FromList> WHERE <Cond>

<Attribute> <SelList> <RelName> <FromList> <Cond> AND <Cond>

B <Attribute> R <RelName> S

<Attr> <Op> <Const>
R.A = "c"

<Attr> <Op> <Attr>
R.C = S.C
```

Select B,D
From R,S
Where R.A = "c" \(\land\) R.C = S.C
Along with Parsing ...

- Semantic checks
  - Do the projected attributes exist in the relations in the From clause?
  - Ambiguous attributes?
  - Type checking, ex: R.A > 17.5

- Expand views
Query rewriting

parse

parse tree

SQL query

Query rewriting

logical query plan

statistics

Physical plan generation

physical query plan

execute

result

Initial logical plan

Rewrite rules

“Best” logical plan

Logical plan
Initial Logical Plan

Relational Algebra: \[ \Pi_{B,D} [ \sigma_{R.A = \text{“c”}} \land R.C = S.C} (RXS) \]
Apply Rewrite Rule (1)

\[ \pi_{B,D} \left( \sigma_{R.A = "c" \land R.C = S.C} \left( R \times S \right) \right) \]

\[ \Pi_{B,D} \left[ \sigma_{R.C = S.C} \left[ \sigma_{R.A = "c"} (R \times S) \right] \right] \]
Apply Rewrite Rule (2)

\[ \pi_{B,D} \sigma_{R.C = S.C} \]

\[ \sigma_{R.A = “c”} X \]

\[ R \quad S \]

\[ \Pi_{B,D} [ \sigma_{R.C=S.C} [\sigma_{R.A=“c”}(R)] X S] \]
Apply Rewrite Rule (3)

\[ \pi_{B,D} \left[ \sigma_{R.C = S.C} \right] \]  \rightarrow  \left[ \sigma_{R.A = "c"} (R) \right] \bowtie S \]

\[ \pi_{B,D} \left[ \sigma_{R.C = S.C} \right] \]  \rightarrow  \left[ \sigma_{R.A = "c"} (R) \right] \bowtie S \]

Natural join
Some Query Rewrite Rules

• Transform one **logical plan** into another
  – Do not use statistics
• Equivalences in relational algebra
• Push-down predicates
• Do projects early
• Avoid cross-products if possible
Equivalences in Relational Algebra

\[ R \bowtie S = S \bowtie R \quad \text{Commutativity} \]
\[ (R \bowtie S) \bowtie T = R \bowtie (S \bowtie T) \quad \text{Associativity} \]

Also holds for: Cross Products, Union, Intersection

\[ R \times S = S \times R \]
\[ (R \times S) \times T = R \times (S \times T) \]
\[ R \cup S = S \cup R \]
\[ R \cup (S \cup T) = (R \cup S) \cup T \]
Apply Rewrite Rule (1)

\[ \pi_{B,D} \left( \sigma_{R.A = "c" \land R.C = S.C} (R \times S) \right) \]

\[ \pi_{B,D} \left( \sigma_{R.C = S.C} \right) \]

\[ \Pi_{B,D} [ \sigma_{R.C = S.C} [\sigma_{R.A = "c"}(R \times S)]] \]
Rules: Project

Let: $X = \text{set of attributes}$

$Y = \text{set of attributes}$

$XY = X \cup Y$

$$\pi_{xy}(R) = \pi_x[\pi_y(R)]$$
Rules: $\sigma + \bowtie$ combined

Let $p =$ predicate with only $R$ attribs
   $q =$ predicate with only $S$ attribs
   $m =$ predicate with only $R,S$ attribs

\[
\sigma_p (R \bowtie S) = [\sigma_p (R)] \bowtie S \\
\sigma_q (R \bowtie S) = R \bowtie [\sigma_q (S)]
\]
Rules: $\sigma + \bowtie$ combined (continued)

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

$\sigma_{p \land q \land m} (R \bowtie S) = \sigma_m [(\sigma_p R) \bowtie (\sigma_q S)]$

$\sigma_{p v q} (R \bowtie S) = [\sigma_p R] \bowtie S \cup [R \bowtie (\sigma_q S)]$
Which are “good” transformations?

- \( \sigma_{p_1 \land p_2}(R) \rightarrow \sigma_{p_1} [\sigma_{p_2}(R)] \)
- \( \sigma_p(R \bowtie S) \rightarrow [\sigma_p(R)] \bowtie S \)
- \( R \bowtie S \rightarrow S \bowtie R \)
- \( \pi_x [\sigma_p(R)] \rightarrow \pi_x \{\sigma_p[\pi_{xz}(R)]\} \)
Conventional wisdom: do projects early

Example: $R(A,B,C,D,E)$

$P: (A=3) \land (B=\text{“cat”})$

$$\pi_E \{\sigma_p (R)\} \; \text{vs.} \; \pi_E \{\sigma_p(\pi_{ABE}(R))\}$$
But: What if we have A, B indexes?

B = “cat”

A = 3

Intersect pointers to get pointers to matching tuples
Bottom line:

• No transformation is always good
• Some are usually good:
  – Push selections down
  – Avoid cross-products if possible
  – Subqueries $\rightarrow$ Joins
Avoid Cross Products (if possible)

Select B,D
From R,S,T,U
Where R.A = S.B ∧
R.C = T.C ∧ R.D = U.D

• Which join trees avoid cross-products?
• If you can't avoid cross products, perform them as late as possible
More Query Rewrite Rules

- Transform one logical plan into another
  - Do not use statistics
- Equivalences in relational algebra
- Push-down predicates
- Do projects early
- Avoid cross-products if possible
- Use left-deep trees
- Subqueries $\rightarrow$ Joins
- Use of constraints, e.g., uniqueness
Query rewriting

parse

parse tree

SQL query

Query rewriting

statistics

Best logical query plan

Physical plan generation

Best physical query plan

execute

result
Physical Plan Generation

Best logical plan

π_{B,D}

σ_{R.A = "c"}

Natural join

R

S

π_{B,D}

σ_{R.A = "c"}

Natural join

R

S

Project

Hash join

Index scan

Table scan

Best logical plan
Query rewriting

- SQL query
  - parse
    - parse tree

Query rewriting

- parse
  - parse tree

Query rewriting

- Best logical query plan
  - Best logical query plan

Physical plan generation

- Best physical query plan
  - Best physical query plan

Physical plan generation

- execute
  - execute
  - result

Enumerate possible physical plans

- Find the cost of each plan

Pick plan with minimum cost

- Pick plan with minimum cost
Physical Plan Generation

Logical Query Plan

P1
C1

P2
C2

....

Pn
Cn

Pick minimum cost one

Physical plans

Costs
Operator Plumbing

• **Materialization**: output of one operator written to disk, next operator reads from the disk

• **Pipelining**: output of one operator directly fed to next operator
Materialization

Materialized here

\( \pi_{B,D} \)

\( \sigma_{R.A = "c"} \)

\( S \)

\( R \)
Iterators: Pipelining

Each operator supports:

- Open()
- GetNext()
- Close()
Iterator for Table Scan (R)

Open() {
    /** initialize variables */
    b = first block of R;
    t = first tuple in block b;
}

GetNext() {
    IF (t is past last tuple in block b) {
        set b to next block;
        IF (there is no next block)
            /** no more tuples */
            RETURN EOT;
        ELSE t = first tuple in b;
    }
    /** return current tuple */
    oldt = t;
    set t to next tuple in block b;
    RETURN oldt;
}

Close() {
    /** nothing to be done */
}
$$\sigma_{R.A = \text{“c”}}$$

**Open()**

```c
/** initialize child */
Child.Open();
```

**Close()**

```c
/** inform child */
Child.Close();
```

**GetNext()**

```c
LOOP:
t = Child.GetNext();
IF (t == EOT) {
    /** no more tuples */
    RETURN EOT;
}
ELSE IF (t.A == "c")
    RETURN t;
ENDLOOP:
```
Open() {
    /** Bulk of the work is here */
    Child.Open();
    Read all tuples from Child and sort them
}

GetNext() {
    IF (more tuples)
        RETURN next tuple in order;
    ELSE RETURN EOT;
}

Close() {
    /** inform child */
    Child.Close();
}
Iterator for Tuple Nested Loop Join

• TNLJ (conceptually)
  for each \( r \in \text{Lexp} \) do
    for each \( s \in \text{Rexp} \) do
      if \( \text{Lexp.C} = \text{Rexp.C} \), output \( r, s \)
Example 1: Left-Deep Plan

Question: What is the sequence of getNext() calls?
Example 2: Right-Deep Plan

Question: What is the sequence of getNext() calls?
Cost Measure for a Physical Plan

• There are many cost measures
  – Time to completion
  – Number of I/Os (we will see a lot of this)
  – Number of getNext() calls

• Tradeoff: Simplicity of estimation Vs. Accurate estimation of performance as seen by user