Introduction to Data Management
CSE 344

Lecture 9: Query Evaluation
Outline

• Lecture 8 review and examples
• Translating SQL to relational algebra
• From logical to physical query plans
• Query execution
Review on the Board

• Client-server architecture
• Life of a query and role of relational algebra
• Relational algebra
  – Relational algebra operators
  – Extended relational algebra operators
Relational Algebra (1/3)

Five basic operators:

- **Union** $(\cup)$ and **Set difference** $(\neg)$
- **Selection**: $\sigma_{\text{condition}}(S)$
  - Condition is Boolean combination $(\land, \lor)$ of terms
  - Term is: attribute op constant, attr. op attr.
  - Op is: $<, \leq, =, \neq, \geq, >$
- **Projection**: $\pi_{\text{list-of-attributes}}(S)$
- **Cross-product** or **cartesian product** $(\times)$
Derived or auxiliary operators:

- **Intersection** ($\cap$), **Division** ($R/S$)
- **Join**: $R \bowtie_\theta S = \sigma_\theta(R \times S)$
- **Variations of joins**
  - Natural, equijoin, theta-join
  - Outer join and semi-join
- **Rename** $\rho_{B_1, \ldots, B_n}(S)$
Relational Algebra (3/3)

Extensions for bags

- **Duplicate elimination**: $\delta$
- **Group by**: $\gamma$ [Same symbol as aggregation]
  - Partitions tuples of a relation into “groups”
- **Sorting**: $\tau$

Other extensions

- **Aggregation**: $\gamma$ (min, max, sum, average, count)
Some Examples

Supplier(sno, sname, scity, sstate)
Part(pno, pname, psize, pcolor)
Supply(sno, pno, qty, price)

Q2: Name of supplier of parts with size greater than 10
\[ \pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{\text{psize}>10} (\text{Part}))) \]

Q3: Name of supplier of red parts or parts with size greater than 10
\[ \pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{\text{psize}>10} (\text{Part}) \cup \sigma_{\text{pcolor}='red'} (\text{Part}))) \]
RA Expressions v.s. Programs

• An Algebra Expression is like a program
  – Several operations
  – Strictly specified order

• But Algebra expressions have limitations
RA and Transitive Closure

- Cannot compute “transitive closure”

<table>
<thead>
<tr>
<th>Name1</th>
<th>Name2</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>Mary</td>
<td>Father</td>
</tr>
<tr>
<td>Mary</td>
<td>Joe</td>
<td>Cousin</td>
</tr>
<tr>
<td>Mary</td>
<td>Bill</td>
<td>Spouse</td>
</tr>
<tr>
<td>Nancy</td>
<td>Lou</td>
<td>Sister</td>
</tr>
</tbody>
</table>

- Find all direct and indirect relatives of Fred
- Cannot express in RA !!! Need to write Java program
Benefits of Relational Algebra

• Physical data independence
  – Can change physical layout of data on disk without changing the relational algebra expression (including adding indexes)

• Can perform query optimization
  – Many relational algebra expressions are equivalent
  – Can choose from many possible implementations

• Easy to parallelize (we will see later)
Outline

• Lecture 8 review and examples
• Translating SQL to relational algebra
• From logical to physical query plans
• Query execution
From SQL to RA

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
    x.price > 100 and z.city = 'Seattle'
From SQL to RA

```
pid = pid

price > 100 and city = 'Seattle'

cid = cid

Product

Purchase

Customer

\[ \Pi \ x.name, z.name \]

\[ \delta \]

\[ \sigma \ price > 100 \text{ and } city = 'Seattle' \]
An Equivalent Expression

Query optimization = finding cheaper equivalent expressions

Diagram:

- Product
  - Purchase
    - Customer
    - δ
      - π
        - x.name, z.name
          - cid = cid
            - σ
              - price > 100
                - pid = pid
                    - σ
                      - city = ‘Seattle’
Operators on Bags

- Duplicate elimination $\delta$
- Grouping $\gamma$
- Sorting $\tau$
Logical Query Plan

\[
\begin{align*}
\text{SELECT} & \quad \text{city, count(*)} \\
\text{FROM} & \quad \text{sales} \\
\text{GROUP BY} & \quad \text{city} \\
\text{HAVING} & \quad \text{sum(price) > 100}
\end{align*}
\]

\[\begin{array}{c}
T1, T2, T3 = \text{temporary tables} \\
\text{sales(product, city, price)}
\end{array}\]
Typical Plan for Block (1/2)

...\[\pi\] fields
\[\sigma\] selection condition
join condition

\{SELECT-PROJECT-JOIN Query\}

R \quad S
Typical Plan For Block (2/2)

\[ \pi \text{ fields} \]
\[ \sigma \text{ selection condition} \]
\[ \gamma \text{ fields, sum/count/min/max(fields)} \]
\[ \text{having}_{\text{condition}} \]
\[ \cdots \]
\[ \cdots \]
How about Subqueries?

```
SELECT  Q.name
FROM    Person Q
WHERE   Q.age > 25
        and not exists
              SELECT *
              FROM Purchase P
              WHERE P.buyer = Q.name
              and P.price > 100
```
How about Subqueries?

```
SELECT Q.name
FROM Person Q
WHERE Q.age > 25
    and not exists
        SELECT *
        FROM Purchase P
        WHERE P.buyer = Q.name
            and P.price > 100
```
Outline

• Lecture 8 review and examples
• Translating SQL to relational algebra
• From logical to physical query plans
• Query execution
Example

Supplier(sid, sname, scity, sstate)  
Supply(sid, pno, quantity)

\[
\text{SELECT sname} \\
\text{FROM Supplier x, Supply y} \\
\text{WHERE x.sid = y.sid} \\
\text{and y.pno = 2} \\
\text{and x.scity} = \text{‘Seattle’} \\
\text{and x.sstate} = \text{‘WA’}
\]

Give a relational algebra expression for this query
Relational Algebra

\[ \pi \text{ sname} \left( \sigma \text{ scity= ‘Seattle’ } \land \text{ sstate= ‘WA’ } \land \text{ pno=2} \left( \text{ Supplier } \bowtie \text{ Supply (sid = sid)} \right) \right) \]

Give ANOTHER relational algebra expression for this query
Relational Algebra

\[
\text{Supplier}(\text{sid, sname, scity, sstate})
\]
\[
\text{Supply}(\text{sid, pno, quantity})
\]

Relational algebra expression is also called the "logical query plan"
A physical query plan is a logical query plan annotated with physical implementation details.

Physical Query Plan 1

(On the fly) \( \pi \text{ sname} \)

(On the fly) \( \sigma \ scity='Seattle' \land sstate='WA' \land pno=2 \)

(Block-nested loop) sid = sid

Supplier (File scan) ---- Supply (File scan)
Physical Query Plan 2

(On the fly) \[ \pi_{\text{sname}} \]

(Sort-merge join) \[ \text{sid} = \text{sid} \]

(Scan write to T1) \[ (a) \ \sigma_{\text{scity}=\text{Seattle} \land \text{sstate}=\text{WA}} \]

(Scan write to T2) \[ (b) \ \sigma_{\text{pno}=2} \]

Supplier (File scan)

Supply (File scan)
Physical Query Plan 3

(On the fly)  (d) $\pi_{sname}$

(On the fly)  

(On the fly)  (c) $\sigma_{\text{scity}=\text{Seattle} \land \text{sstate}=\text{WA}}$

(Use index)  

(a) $\sigma_{pno=2}$

Supply  

(Index lookup on pno)  

Assume: clustered

Supplier  

(Index lookup on sid)  

Doesn’t matter if clustered or not
Query Evaluation Steps

1. Parse & Check Query
2. Select Logical Plan
3. Select Physical Plan
4. Query Execution

SQL query

Query optimization

Logical plan

Physical plan

Disk
Outline

• Lecture 8 review and examples
• Translating SQL to relational algebra
• From logical to physical query plans
• Query execution
Iterator Interface

• Each operator implements iterator interface
• Interface has only three methods
  • open()
    – Initializes operator state
    – Sets parameters such as selection condition
  • get_next()
    – Operator invokes get_next() recursively on its inputs
    – Performs processing and produces an output tuple
  • close(): cleans-up state
Pipelined Execution

• Applies parent operator to tuples directly as they are produced by child operators

• Benefits
  – No operator synchronization issues
  – Saves cost of writing intermediate data to disk
  – Saves cost of reading intermediate data from disk
  – Good resource utilizations on single processor

• This approach is used whenever possible
Pipelined Execution

(On the fly) $\pi_{sname}$

(On the fly) $\sigma_{sscity='Seattle' \land sstate='WA' \land pno=2}$

(Nested loop) $sno = sno$

Suppliers (File scan)  Supplies (File scan)
Intermediate Tuple 
Materialization

• Writes the results of an operator to an intermediate table on disk

• No direct benefit but
• Necessary for some operator implementations
• When operator needs to examine the same tuples multiple times
Intermediate Tuple Materialization

(On the fly)

(Sort-merge join)

(Scan: write to T1)  (Scan: write to T2)

\[ \sigma_{s\text{state}='WA'} \land s\text{city}='Seattle' \]

\[ \pi_{s\text{name}} \]

Suppliers

(File scan)

Supplies

(File scan)

\[ \sigma_{p\text{no}=2} \]

\[ s\text{no} = s\text{no} \]