Introduction to Data Management
CSE 344

Lecture 9: Relational Algebra and Query Evaluation
Today

• Relational algebra

• Physical plans and query evaluation
Relational Algebra Operators

- Union $\cup$, intersection $\cap$, difference $-$
- Selection $\sigma$
- Projection $\pi$
- Cartesian product $\times$, join $\Join$
- Rename $\rho$
- Duplicate elimination $\delta$
- Grouping and aggregation $\gamma$
- Sorting $\tau$

All operators take in 1 or more relations as inputs and return another relation.
Join Summary

• **Theta-join**: \( R \bowtie_\theta S = \sigma_\theta (R \times S) \)
  – Join of \( R \) and \( S \) with a join condition \( \theta \)
  – Cross-product followed by selection \( \theta \)

• **Equijoin**: \( R \bowtie_\theta S = \pi_A (\sigma_\theta (R \times S)) \)
  – Join condition \( \theta \) consists only of equalities
  – Projection \( \pi_A \) drops all redundant attributes

• **Natural join**: \( R \bowtie S = \pi_A (\sigma_\theta (R \times S)) \)
  – Equality on all fields with same name in \( R \) and in \( S \)
  – Projection \( \pi_A \) drops all redundant attributes
So Which Join Is It?

When we write $R \bowtie S$ we usually mean an equijoin, but we often omit the equality predicate when it is clear from the context.
More Joins

• **Outer join**
  – Include tuples with no matches in the output
  – Use NULL values for missing attributes
  – Does not eliminate duplicate columns

• **Variants**
  – Left outer join
  – Right outer join
  – Full outer join
### Outer Join Example

**AnonPatient P**

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
<td>lung</td>
</tr>
</tbody>
</table>

**AnonJob J**

<table>
<thead>
<tr>
<th>job</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>lawyer</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>cashier</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

**P ⨝ J**

<table>
<thead>
<tr>
<th>P.age</th>
<th>P.zip</th>
<th>disease</th>
<th>job</th>
<th>J.age</th>
<th>J.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>lawyer</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>cashier</td>
<td>20</td>
<td>98120</td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
<td>lung</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
Some Examples

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

Name of supplier of parts with size greater than 10
\( \pi_{\text{sname}} (\text{Supplier} \Join (\text{Supply} \Join (\sigma_{\text{psize}>10} (\text{Part})))) \)

Name of supplier of red parts or parts with size greater than 10
\( \pi_{\text{sname}} (\text{Supplier} \Join (\text{Supply} \Join (\sigma_{\text{psize}>10} (\text{Part}) \cup \sigma_{\text{pcolor}='red'} (\text{Part})))) \)

Can be represented as trees as well (as seen from lecture 7)
Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

From SQL to RA

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'

Diagram:

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

Join:
- δ: x.pid = y.pid
- σ: x.price > 100 and z.city = 'Seattle'
- Π: x.name, z.name
From SQL to RA

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'

Can you think of a “better” plan?
Can you think of a “better” plan?

Query optimization: finding cheaper, equivalent expressions
Extended RA: Operators on Bags

- Duplicate elimination $\delta$
- Grouping $\gamma$
  - Takes in relation and a list of grouping operations (e.g., aggregates). Returns a new relation.
- Sorting $\tau$
  - Takes in a relation, a list of attributes to sort on, and an order. Returns a new relation.
Using Extended RA Operators

\[\begin{align*}
\text{SELECT } & \text{city, count(*)} \\
\text{FROM } & \text{sales} \\
\text{GROUP BY } & \text{city} \\
\text{HAVING } & \text{sum(price) > 100}
\end{align*}\]
Typical Plan for a Query (1/2)

Answer

\[ \pi_{\text{fields}} \]

\[ \sigma_{\text{selection condition}} \]

\[ \text{join condition} \]

\[ \text{join condition} \]

\[ \text{join condition} \]

\[ \ldots \]

\[ \text{SELECT-PROJECT-JOIN Query} \]

\[ \text{SELECT fields} \]

\[ \text{FROM R, S, ...} \]

\[ \text{WHERE condition} \]
Typical Plan for a Query (1/2)

\[ \pi \text{fields}, \sum/\text{count}/\text{min}/\text{max}(\text{fields}) \]

\[ \sigma \text{where condition} \]

\[ \text{JOIN condition} \]

\[ \sigma \text{having condition} \]

\[ \text{SELECT fields FROM R, S, ... WHERE condition GROUP BY fields HAVING condition} \]
How about Subqueries?

```
SELECT  Q.sno
FROM    Supplier Q
WHERE   Q.sstate = 'WA'
        and not exists
        (SELECT *
         FROM  Supply P
         WHERE  P.sno = Q.sno
                and P.price > 100)
```
How about Subqueries?

```
SELECT  Q.sno
FROM    Supplier Q
WHERE   Q.sstate = 'WA'
        and not exists 
        (SELECT * 
         FROM Supply P 
         WHERE P.sno = Q.sno 
            and P.price > 100)
```
How about Subqueries?

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
and not exists
(SELECT *
FROM Supply P
WHERE P.sno = Q.sno
and P.price > 100)
```

De-Correlation

```
SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = 'WA'
and Q.sno not in
(SELECT P.sno
FROM Supply P
WHERE P.price > 100)
```
How about Subqueries?

\[
\begin{align*}
\text{(SELECT } & \text{ Q.sno} \\
\text{ FROM } & \text{ Supplier Q} \\
\text{ WHERE } & \text{ Q.sstate} = \text{‘WA’}) \\
\text{ EXCEPT} \\
(\text{SELECT } & \text{ P.sno} \\
\text{ FROM } & \text{ Supply P} \\
\text{ WHERE } & \text{ P.price} > 100) \\
\text{ EXCEPT} & = \text{ set difference}
\end{align*}
\]

SELECT Q.sno
FROM Supplier Q
WHERE Q.sstate = ‘WA’
EXCEPT
(SELECT P.sno
FROM Supply P
WHERE P.price > 100)
CSE 344 - Winter 2017

How about Subqueries?

\[
\begin{align*}
    \text{Finally...} \\
    \pi_{\text{sno}} \\
    \left( \text{SELECT } Q.\text{sno} \right. \\
    \text{FROM Supplier } Q \\
    \text{WHERE } Q.\text{sstate} = 'WA' \\
    \left. \text{EXCEPT} \right) \\
    \text{(SELECT P.\text{sno}} \\
    \text{FROM Supply } P \\
    \text{WHERE P.\text{price} > 100)}
\end{align*}
\]
From Logical RA Plans to Physical Plans
Query Evaluation Steps Review

SQL query

Parse & Rewrite Query

Select Logical Plan

Select Physical Plan

Query Execution

Disk

Logical plan (RA)

Physical plan

Query optimization
Supplier\((\text{sid, sname, scity, sstate})\)
Supply\((\text{sid, pno, quantity})\)

Relational Algebra

\[
\begin{align*}
\text{SELECT sname} \\
\text{FROM Supplier x, Supply y} \\
\text{WHERE x.sid = y.sid} \\
\quad \text{and } y.pno = 2 \\
\quad \text{and } x.scity = 'Seattle' \\
\quad \text{and } x.sstate = 'WA'
\end{align*}
\]

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

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
  and y.pno = 2
  and x.scity = 'Seattle'
  and x.sstate = 'WA'
```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Query Plan 2

(On the fly)  \( \Pi_{sname} \)

(On the fly)  \( \sigma_{\text{scity}=\text{'Seattle'} \text{ and } \text{sstate}=\text{'WA'} \text{ and } \text{pno}=2} \)

(Hash join)  \( \text{sid = sid} \)

Supplier (File scan)
Supply (File scan)

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
and y.pno = 2
and x.scity = 'Seattle'
and x.sstate = 'WA'

Same logical query plan
Different physical plan

CSE 344 - Winter 2017
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Query Plan 3

Different but equivalent logical query plan; different physical plan

SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'

(a) $\sigma_{\text{scity='Seattle' and sstate='WA'}}$
(b) $\sigma_{\text{pno=}2}$

(c) \( \pi_{\text{sname}} \) (d)

(Scan & write to T2)
(Scan & write to T1)
(Sort-merge join)
(On the fly)
Query Optimization Problem

• For each SQL query… many logical plans

• For each logical plan… many physical plans

• How do find a fast physical plan?
  – Will discuss in a few lectures
  – First we need to understand how query operators are implemented
Query Execution
Iterator Interface for Query Operators

- **open()**
  - Initializes operator state
  - Sets parameters such as selection condition

- **next()**
  - Operator invokes get_next() recursively on its inputs
  - Performs processing and produces an output tuple

- **close()**: clean-up state
Pipelined Query Execution

(On the fly)   \( \prod_{\text{sname}} \)   (On the fly)  \( \sigma_{\text{scity} = 'Seattle' \text{ and } \text{sstate} = 'WA' \text{ and } \text{pno} = 2} \)

(Nested loop)  

\( \text{sno} = \text{sno} \)

(On the fly)  open()

Suppliers  open()
(\text{File scan})

Supplies  open()
(\text{File scan})
Pipelined Query Execution

(On the fly) \[ \Pi_{\text{fname}} \]

(On the fly) \[ \sigma_{\text{scity}=\text{Seattle} \ \text{and} \ sstate=\text{WA} \ \text{and} \ pno=2} \]

(Nested loop) \[ \text{sno} = \text{sno} \]

Suppliers (File scan)

Supplies (File scan)

next()

next()
Pipelined Execution

• Tuples generated by an operator are immediately sent to the parent

• Benefits:
  – No operator synchronization issues
  – No need to buffer tuples between operators
  – Saves cost of writing intermediate data to disk
  – Saves cost of reading intermediate data from disk

• This approach is used whenever possible
Query Execution Bottom Line

- SQL query transformed into **physical plan**
  - **Access path selection** for each relation
    - Scan the relation or use an index (next lecture)
  - **Implementation choice** for each operator
    - Nested loop join, hash join, etc.
  - **Scheduling decisions** for operators
    - Pipelined execution or intermediate materialization
- Pipelined execution of physical plan
Physical Data Independence

- Applications are insulated from changes in physical storage details

- SQL and relational algebra facilitate physical data independence
  - Both languages input and output relations
  - Can choose different implementations for operators