CSE 344

JANUARY 22ND – RELATIONAL ALGEBRA
ASSORTED MINUTIAE

• HW2 and Online Quiz 2 due on Wednesday
• Azure accounts will be created tonight
  • You will be added to your account with your @cs.washington.edu email address
  • If you don’t have one, email me and I will attach a different address
  • When you get access, make sure that you only run queries that you need
  • Due next Friday (some overlap)
TODAY’S LECTURE

• Finalizing Subqueries
• Queries as Relational algebra
MONOTONE QUERIES

Definition A query $Q$ is **monotone** if:

- Whenever we add tuples to one or more input tables, the answer to the query will not lose any of the tuples
MONOTONE QUERIES

Theorem: If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.
MONOTONE QUERIES

**Theorem:** If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.

**Proof.** We use the nested loop semantics: if we insert a tuple in a relation $R_i$, this will not remove any tuples from the answer.
MONOTONE QUERIES

The query:

Find all companies s.t. all their products have price < 200

is not monotone
The query: Find all companies s.t. all their products have price < 200 is not monotone

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>19.99</td>
<td>c001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>c001</td>
<td>Sunworks</td>
<td>Bonn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunworks</td>
</tr>
</tbody>
</table>
The query:

Find all companies s.t. all their products have price < 200 is not monotone

Consequence: If a query is not monotonic, then we cannot write it as a SELECT-FROM-WHERE query without nested subqueries
QUERIES THAT MUST BE NESTED

Queries with universal quantifiers or with negation
QUERIES THAT MUST BE NESTED

Queries with universal quantifiers or with negation

Queries that use aggregates in certain ways

- \texttt{sum(..)} and \texttt{count(*)} are NOT monotone, because they do not satisfy set containment.
- \texttt{select count(*) from R} is not monotone!
GROUP BY V.S. NESTED QUERIES

```
SELECT product, Sum(quantity) AS TotalSales
FROM Purchase
WHERE price > 1
GROUP BY product

SELECT DISTINCT x.product, (SELECT Sum(y.quantity)
 FROM Purchase y
 WHERE x.product = y.product
 AND y.price > 1)
 AS TotalSales
FROM Purchase x
WHERE x.price > 1
```

Why twice?
MORE UNNESTING

Find authors who wrote $\geq 10$ documents:
Find authors who wrote ≥ 10 documents:

Attempt 1: with nested queries

```sql
SELECT DISTINCT Author.name
FROM Author
WHERE (SELECT count(Wrote.url) 
      FROM Wrote 
      WHERE Author.login=Wrote.login) >= 10
```
Find authors who wrote ≥ 10 documents:

Attempt 1: with nested queries

Attempt 2: using GROUP BY and HAVING

```
SELECT Author.name
FROM Author, Wrote
WHERE Author.login=Wrote.login
GROUP BY Author.name
HAVING count(wrote.url) >= 10
```
FINDING WITNESSES

For each city, find the most expensive product made in that city
FINDING WITNESSES

For each city, find the most expensive product made in that city

Finding the maximum price is easy...

```
SELECT  x.city, max(y.price)
FROM    Company x, Product y
WHERE   x.cid = y.cid
GROUP BY x.city;
```

But we need the witnesses, i.e., the products with max price
FINDING WITNESSES

To find the witnesses, compute the maximum price in a subquery (in FROM or in WITH)

WITH CityMax AS
   (SELECT x.city, max(y.price) as maxprice
    FROM Company x, Product y
    WHERE x.cid = y.cid
    GROUP BY x.city)
SELECT DISTINCT u.city, v.pname, v.price
FROM Company u, Product v, CityMax w
WHERE u.cid = v.cid
   and u.city = w.city
   and v.price = w.maxprice;
FINDING WITNESSES

To find the witnesses, compute the maximum price in a subquery (in FROM or in WITH)

```
SELECT DISTINCT u.city, v.pname, v.price
FROM Company u, Product v,
    (SELECT x.city, max(y.price) as maxprice
     FROM Company x, Product y
     WHERE x.cid = y.cid
     GROUP BY x.city) w
WHERE u.cid = v.cid
    and u.city = w.city
    and v.price = w.maxprice;
```
FINDING WITNESSES

Or we can use a subquery in where clause

```
SELECT u.city, v.pname, v.price
FROM Company u, Product v
WHERE u.cid = v.cid
  and v.price >= ALL (SELECT y.price
                       FROM Company x, Product y
                       WHERE u.city=x.city
                       and x.cid=y.cid);
```
FINDING WITNESSES

There is a more concise solution here:

```
SELECT u.city, v.pname, v.price
FROM Company u, Product v, Company x, Product y
WHERE u.cid = v.cid and u.city = x.city
and x.cid = y.cid
GROUP BY u.city, v.pname, v.price
HAVING v.price = max(y.price)
```
RELATIONAL ALGEBRA

Set-at-a-time algebra, which manipulates relations

In SQL we say *what* we want

In RA we can express *how* to get it

Every DBMS implementations converts a SQL query to RA in order to execute it

An RA expression is called a *query plan*
BASICS

• Relations and attributes
• Functions that are applied to relations
  – Return relations
  – Can be composed together
  – Often displayed using a tree rather than linearly
  – Use Greek symbols: $\sigma$, $\pi$, $\delta$, etc
SETS V.S. BAGS

Sets: \{a, b, c\}, \{a, d, e, f\}, \{\}\ldots

Bags: \{a, a, b, c\}, \{b, b, b, b, b\}\ldots

Relational Algebra has two flavors:
Set semantics = standard Relational Algebra
Bag semantics = extended Relational Algebra

DB systems implement bag semantics (Why?)
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
UNION AND DIFFERENCE

\[ R_1 \cup R_2 \]
\[ R_1 - R_2 \]

Only make sense if R1, R2 have the same schema

What do they mean over bags?
WHAT ABOUT INTERSECTION?

Derived operator using minus

\[ R_1 \cap R_2 = R_1 - (R_1 - R_2) \]

Derived using join

\[ R_1 \cap R_2 = R_1 \bowtie R_2 \]
SELECTION

Returns all tuples which satisfy a condition

Examples

- $\sigma_{\text{Salary} > 40000}$ (Employee)
- $\sigma_{\text{name} = \text{“Smith”}}$ (Employee)

The condition c can be $=, <, \leq, >, \geq, <$ combined with AND, OR, NOT
### Employee

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>50000</td>
</tr>
</tbody>
</table>

\[\sigma_{\text{Salary} > 40000} \text{ (Employee)}\]

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>5423341</td>
<td>Smith</td>
<td>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>Fred</td>
<td>50000</td>
</tr>
</tbody>
</table>
PROJECTION

Eliminates columns

\[ \pi_{A_1, \ldots, A_n}(R) \]

Example: project social-security number and names:

\[ \pi_{\text{SSN, Name}}(\text{Employee}) \rightarrow \text{Answer(SSN, Name)} \]

Different semantics over sets or bags! Why?
### Employee

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234545</td>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>5423341</td>
<td>John</td>
<td>60000</td>
</tr>
<tr>
<td>4352342</td>
<td>John</td>
<td>20000</td>
</tr>
</tbody>
</table>

\[ \pi_{\text{Name, Salary}} (\text{Employee}) \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>John</td>
<td>60000</td>
</tr>
<tr>
<td>John</td>
<td>20000</td>
</tr>
</tbody>
</table>

Bag semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>20000</td>
</tr>
<tr>
<td>John</td>
<td>60000</td>
</tr>
</tbody>
</table>

Set semantics

Which is more efficient?
<table>
<thead>
<tr>
<th>no</th>
<th>name</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>p4</td>
<td>98120</td>
<td>heart</td>
</tr>
<tr>
<td>2</td>
<td>p2</td>
<td>98125</td>
<td>heart</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{disease} = 'heart'} (\text{Patient}) \]

\[ \pi_{\text{zip}, \text{disease}} (\sigma_{\text{disease} = 'heart'} (\text{Patient})) \]

<table>
<thead>
<tr>
<th>no</th>
<th>name</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>p4</td>
<td>98120</td>
<td>heart</td>
</tr>
<tr>
<td>3</td>
<td>p3</td>
<td>98120</td>
<td>lung</td>
</tr>
<tr>
<td>2</td>
<td>p2</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>1</td>
<td>p1</td>
<td>98125</td>
<td>flu</td>
</tr>
</tbody>
</table>

\[ \pi_{\text{zip}, \text{disease}} (\sigma_{\text{disease} = 'heart'} (\text{Patient})) \]

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

\[ \text{Patient} \]
CARTESIAN PRODUCT

Each tuple in R1 with each tuple in R2

$\text{R1} \times \text{R2}$

Rare in practice; mainly used to express joins
# CROSS-PRODUCT EXAMPLE

<table>
<thead>
<tr>
<th>Employee</th>
<th>Dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>SSN</td>
</tr>
<tr>
<td>John</td>
<td>9999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
</tr>
</tbody>
</table>

## Employee X Dependent

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>9999999999</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>John</td>
<td>9999999999</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>9999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>7777777777</td>
<td>7777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>
NATURAL JOIN

\[
R1 \bowtie R2
\]

Meaning: \( R1 \bowtie R2 = \Pi_A (\sigma_\theta (R1 \times R2)) \)

Where:

- Selection \( \sigma_\theta \) checks equality of all common attributes (i.e., attributes with same names)
- Projection \( \Pi_A \) eliminates duplicate common attributes
# Natural Join Example

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
</tr>
</tbody>
</table>

\[
R \bowtie S = \Pi_{ABC}(\sigma_{R.B=S.B}(R \times S))
\]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>X</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
<td>U</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
<td>V</td>
</tr>
<tr>
<td>Z</td>
<td>V</td>
<td>W</td>
</tr>
</tbody>
</table>
## Natural Join

### Example 2

<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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>Bob</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

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>
</tbody>
</table>

Voters $V$  

$P \bowtie V$

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>Alice</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>Bob</td>
</tr>
</tbody>
</table>
THETA JOIN

A join that involves a predicate

\[ R_1 \bowtie_\theta R_2 = \sigma_\theta (R_1 \times R_2) \]

Here \( \theta \) can be any condition

No projection in this case!

For our voters/patients example:

\[ P \bowtie \quad P.zip = V.zip \text{ and } P.age >= V.age - 1 \text{ and } P.age <= V.age + 1 \]
EQUIJOIN

A theta join where $\theta$ is an equality predicate

\[ R1 \bowtie_{\theta} R2 = \sigma_{\theta} (R1 \times R2) \]

By far the most used variant of join in practice

What is the relationship with natural join?
**EQUIJOIN 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>
</tbody>
</table>

Voters V

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

P \( owtie_{\text{age}=\text{V.age}} \) V

<table>
<thead>
<tr>
<th>P.age</th>
<th>P.zip</th>
<th>P.disease</th>
<th>V.name</th>
<th>V.age</th>
<th>V.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>98125</td>
<td>heart</td>
<td>p1</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>
**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 $\theta$
- No projection

**Equijoin:** $R \bowtie_{\theta} S = \sigma_{\theta} (R \times S)$
- Join condition $\theta$ consists only of equalities
- No projection

**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>

<table>
<thead>
<tr>
<th>P.age</th>
<th>P.zip</th>
<th>P.disease</th>
<th>J.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
\[ \pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{\text{psize}>10} \text{(Part)}) \]

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

Can be represented as trees as well
REPRESNETING RA QUERIES AS TREES

\[
\begin{align*}
\text{Supplier} & (sno, sname, scity, sstate) \\
\text{Part} & (pno, pname, psize, pcolor) \\
\text{Supply} & (sno, pno, qty, price)
\end{align*}
\]

\[
\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10} (\text{Part})))
\]

\[
\sigma_{psize>10}
\]

Answer

Supplier

Part

Supply
RELATIONAL ALGEBRA OPERATORS

- Union $\cup$
- Intersection $\cap$
- Difference $\cdot$
- 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.
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

```
SELECT city, sum(quantity) 
FROM sales 
GROUP BY city 
HAVING count(*) > 100
```

\[ \Pi_{\text{city, } q} \]
\[ \sigma_{c > 100} \]
\[ \gamma_{\text{city, sum(quantity)} \rightarrow q, \text{count(*)} \rightarrow c} \]
\[ \text{sales(product, city, quantity)} \]

\( T_1, T_2 = \text{temporary tables} \)
TYPICAL PLAN FOR A QUERY (1/2)

Answer

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

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

\[ \text{join condition} \]

\[ \text{join condition} \]

\[ \text{join condition} \]

\[ \text{join condition} \]

\[ \text{R} \]

\[ \text{S} \]

\{ SELECT fields
FROM R, S, ...
WHERE condition \}

\{ SELECT-PROJECT-JOIN
Query \}
TYPICAL PLAN FOR A QUERY (1/2)

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

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

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

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

\[ \text{join condition} \]

\[ \text{…} \]

\[ \text{…} \]

\[ \text{SELECT fields} \]
\[ \text{FROM R, S, …} \]
\[ \text{WHERE condition} \]
\[ \text{GROUP BY fields} \]
\[ \text{HAVING condition} \]
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?

```sql
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)
```

The diagram illustrates the process of de-correlation, which simplifies the query by removing the subquery from the `WHERE` clause.

```
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?

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

\text{EXCEPT} = \text{set difference}
HOW ABOUT SUBQUERIES?

Finally…

\[
\begin{align*}
\text{SELECT } & Q.sno \\
\text{FROM } & \text{Supplier } Q \\
\text{WHERE } & Q.sstate = 'WA' \\
\text{EXCEPT} & \\
\text{SELECT } & P.sno \\
\text{FROM } & \text{Supply } P \\
\text{WHERE } & P.price > 100 \\
\end{align*}
\]

\[
\begin{align*}
\pi_{\text{sno}} & \rightarrow \text{Supplier} \\
\sigma_{\text{sstate}=\text{'WA'}} & \rightarrow \text{Part} \\
\sigma_{\text{Price} > 100} & \rightarrow \text{Supply} \\
\end{align*}
\]
SUMMARY OF RA AND SQL

SQL = a declarative language where we say *what* data we want to retrieve

RA = an algebra where we say *how* we want to retrieve the data

Theorem: SQL and RA can express exactly the same class of queries

RDBMS translate SQL → RA, then optimize RA
SQL (and RA) cannot express ALL queries that we could write in, say, Java

Example:

- Parent(p,c): find all descendants of ‘Alice’
- No RA query can compute this!
- This is called a recursive query

Next lecture: Datalog is an extension that can compute recursive queries