Announcements

• Lecture on Wed. Jan 17 – CANCELED
  Makeup: Tue. Jan 16, 10am-11:20am, CSE 305

• Reading assignments are posted: first due on Jan 16

• Project milestones posted: first due this Friday

• Homework 1 due next Friday

• Discussion board is up: say “hello” there!
Outline

Two topics today

- Crash course in SQL
- Relational algebra
Structured Query Language: SQL

• Influenced by relational calculus (= First Order Logic)

• SQL is a declarative query language
  – We say what we want to get
  – We don’t say how we should get it

• SQL has many parts
  – Data definition language (DDL)
  – Data manipulation language (DML)
  – ...

Outline

• You study independently SQL DDL
  – Data Definition Language
  – CREATE TABLE, DROP TABLE, CREATE INDEX, CLUSTER, ALTER TABLE, …
  – E.g. google for the postgres manual, or type this in psql:
    \h create
    \h create table
    \h cluster

• Today: crash course in SQL DML
  – Data Manipulation Language
  – SELECT-FROM-WHERE-GROUPBY
  – Study independently: INSERT/DELETE/MODIFY
SQL Query

Basic form:

```
SELECT <attributes>
FROM <one or more relations>
WHERE <conditions>
```
Simple SQL Query

```
SELECT PName, Price, Manufacturer
FROM Product
WHERE Price > 100
```

```
PName | Price  | Category | Manufacturer
-----|--------|----------|---------------
Gizmo | $19.99 | Gadgets  | GizmoWorks
Powergizmo | $29.99 | Gadgets  | GizmoWorks
SingleTouch | $149.99 | Photography | Canon
MultiTouch | $203.99 | Household | Hitachi
```

“selection” and “projection”
Eliminating Duplicates

SELECT DISTINCT category
FROM Product

Compare to:

SELECT category
FROM Product

Category
Gadgets
Photography
Household
Ordering the Results

```sql
SELECT  pname, price, manufacturer
FROM    Product
WHERE   category='gizmo' AND price > 50
ORDER BY price, pname
```

Ascending, unless you specify the DESC keyword.
Can also request only top-k with LIMIT clause

```sql
SELECT  pname, price, manufacturer
FROM    Product
WHERE   category='gizmo' AND price > 50
ORDER BY price, pname
LIMIT 10
```
Joins

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all products under $200 manufactured in Japan; return their names and prices.

```
SELECT P.pname, P.price
FROM Product P, Company C
WHERE P.manufacturer=C.cname AND C.country='Japan'
    AND P.price <= 200
```

```
SELECT P.pname, P.price
FROM Product P JOIN Company C ON P.manufacturer=C.cname
WHERE C.country='Japan' AND P.price <= 200
```
Joins

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all countries that manufacture products in both the gadget category and in the photography category

[in class, or at home]
Semantics of SQL Queries

```sql
SELECT a_1, a_2, ..., a_k
FROM R_1 AS x_1, R_2 AS x_2, ..., R_n AS x_n
WHERE Conditions
```

```python
Answer = {}
for x_1 in R_1 do
    for x_2 in R_2 do
        ...
        for x_n in R_n do
            if Conditions
                then Answer = Answer ∪ {(a_1,...,a_k)}
return Answer
```
Aggregation

```
SELECT avg(price)
FROM Product
WHERE maker='Toyota'
```

```
SELECT count(*)
FROM Product
WHERE year > 1995
```

SQL supports several aggregation operations:

- sum, count, min, max, avg

Except count, all aggregations apply to a single attribute
Grouping and Aggregation

Purchase(product, price, quantity)

Find total quantities for all sales over $1, by product.

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

Let’s see what this means…
Grouping and Aggregation

1. Compute the FROM and WHERE clauses.

2. Group by the attributes in the GROUPBY clause.

3. Compute the SELECT clause: grouped attributes and aggregates.
### 1&2. FROM-WHERE-GROUPBY

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Bagel</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>0.5</td>
<td>50</td>
</tr>
<tr>
<td>Banana</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

WHERE price > 1
3. SELECT

What can go in SELECT clause? Will return ONE TUPLE per group

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Bagel</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>0.5</td>
<td>50</td>
</tr>
<tr>
<td>Banana</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

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

Same query as earlier, except that we consider only products that had at least 30 sales.

```
SELECT product, sum(price*quantity)
FROM Purchase
WHERE price > 1
GROUP BY product
HAVING Sum(quantity) > 30
```

HAVING clause contains conditions on aggregates.
WHERE vs HAVING

• WHERE condition is applied to individual rows
  – The rows may or may not contribute to the aggregate
  – No aggregates allowed here

• HAVING condition is applied to the entire group
  – Entire group is returned, or not at all
  – May use aggregate functions in the group
General form of Grouping and Aggregation

SELECT S
FROM R₁, ..., Rₙ
WHERE C₁
GROUP BY a₁, ..., aₖ
HAVING C₂

S = may contain attributes a₁, ..., aₖ and/or any aggregates but NO OTHER ATTRIBUTES
C₁ = is any condition on the attributes in R₁, ..., Rₙ
C₂ = is any condition on aggregate expressions and on attributes a₁, ..., aₖ
Semantics of SQL With Group-By

```
SELECT S
FROM R_1, ..., R_n
WHERE C1
GROUP BY a_1, ..., a_k
HAVING C2
```

Evaluation steps:
1. Evaluate FROM-WHERE using Nested Loop Semantics
2. Group by the attributes $a_1, ..., a_k$
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result
Subqueries

- A subquery is a SQL query nested inside a larger query
- Such inner-outer queries are called nested queries
- A subquery may occur in:
  - A SELECT clause
  - A FROM clause
  - A WHERE clause

- Rule of thumb: avoid writing nested queries when possible; keep in mind that sometimes it’s impossible
Subqueries in WHERE

Product (pname, price, cid)
Company(cid, cname, city)

Find all companies that make some products with price < 200

Using EXISTS:

```
SELECT DISTINCT  C.cname
FROM      Company C
WHERE EXISTS (SELECT *
               FROM  Product P
               WHERE C.cid = P.cid and P.price < 200)
```
Subqueries in WHERE

Product (pname, price, cid)
Company(cid, cname, city)

Find all companies that make some products with price < 200

Using **IN**

```
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
                 FROM Product P
                 WHERE P.price < 200)
```
Subqueries in WHERE

Find all companies that make some products with price < 200

Using ANY:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 > ANY (SELECT price
FROM Product P
WHERE P.cid = C.cid)
```
Subqueries in WHERE

Find all companies that make some products with price < 200

Now let's unnest it:

```sql
SELECT DISTINCT C.cname
FROM Company C, Product P
WHERE C.cid= P.cid and P.price < 200
```

Existential quantifiers are easy! 😊
Subqueries in WHERE

Product (pname, price, cid)
Company(cid, cname, city)

Find all companies that make only products with price < 200

same as:
Find all companies whose products all have price < 200

Universal quantifiers are hard! 😞
Subqueries in WHERE

1. Find the other companies: i.e. s.t. some product $\geq 200$

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid IN (SELECT P.cid
                    FROM Product P
                    WHERE P.price $\geq$ 200)
```

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

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE C.cid NOT IN (SELECT P.cid
                     FROM Product P
                     WHERE P.price $\geq$ 200)
```
Subqueries in WHERE

Product (pname, price, cid)
Company(cid, cname, city)

Find all companies that make only products with price < 200

Using EXISTS:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE NOT EXISTS (SELECT *
                  FROM Product P
                  WHERE P.cid = C.cid and P.price >= 200)
```
Subqueries in WHERE

Product (pname, price, cid)
Company(cid, cname, city)

Find all companies that make only products with price < 200

Using ALL:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 > ALL (SELECT price
FROM Product P
WHERE P.cid = C.cid)
```
Can we unnest the *universal quantifier* query?

- **Definition**: A query Q is **monotone** if:
  - Whenever we add tuples to one or more of the tables…
  - … the answer to the query cannot contain fewer tuples

- **Fact**: all unnested queries are monotone
  - Proof: using the “nested for loops” semantics

- **Fact**: Query with universal quantifier is not monotone

- **Consequence**: we cannot unnest a query with a universal quantifier
More SQL

Things you need to learn on your own (e.g. read the slides from CSE344):

• Three valued logic of SQL: false, unknown, true

• Aggregating over empty groups using left outer join

• How to express argmax in SQL
Outline

Two topics today

• Crash course in SQL

• Relational algebra
Relational Algebra

• Simple algebra over relations: selection, projection, join, union, difference

• Unlike SQL, RA specifies in which order to perform operations; used to compile and optimize SQL

• Declarative? Mostly yes, because we still don’t specify (yet) how each RA operator is to be executed
Relational Operators

- **Selection**: $\sigma_{\text{condition}}(S)$
- **Projection**: $\pi_{\text{list-of-attributes}}(S)$
- **Union** ($\cup$)
- **Set difference** ($-$),
- **Cross-product or cartesian product** ($\times$)
- **Join**: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
- **Intersection** ($\cap$)
- **Division**: $R/S$
- **Rename**: $\rho(R(F), E)$

*Note: both set and bag semantics!*
### Selection & Projection Examples

**Patient**

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

\[
\pi_{\text{zip},\text{disease}}(\text{Patient})
\]

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

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

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

\[
\pi_{\text{zip}}(\sigma_{\text{disease}=\text{‘heart’}}(\text{Patient}))
\]

<table>
<thead>
<tr>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>98120</td>
</tr>
<tr>
<td>98125</td>
</tr>
</tbody>
</table>
Cross-Product 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>

### $P \times V$

<table>
<thead>
<tr>
<th>P.age</th>
<th>P.zip</th>
<th>disease</th>
<th>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>54</td>
<td>98125</td>
<td>heart</td>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</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 Galore

- **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**: \( R \bowtie_\theta S = \sigma_\theta(R \times S) \)
  - Theta-join where \( \theta \) consists only of equalities

- **Natural join**: \( R \bowtie S = \pi_A (\sigma_\theta(R \times S)) \)
  - Equijoin on attributes with the same name
  - Followed by removal (projection) of duplicate attributes
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>
<tr>
<td>p3</td>
<td>20</td>
<td>98123</td>
</tr>
</tbody>
</table>

\[
P \bowtie_{P.\text{age}=V.\text{age}} V
\]

<table>
<thead>
<tr>
<th>P.\text{age}</th>
<th>P.\text{zip}</th>
<th>P.\text{disease}</th>
<th>V.\text{name}</th>
<th>V.\text{age}</th>
<th>V.\text{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>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>p3</td>
<td>20</td>
<td>98123</td>
</tr>
</tbody>
</table>
## Theta-Join Example

### AnonPatient P

<table>
<thead>
<tr>
<th>age</th>
<th>zip</th>
<th>disease</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>98125</td>
<td>heart</td>
</tr>
<tr>
<td>19</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 ⋈ P.zip = V.zip and P.age <= V.age + 1 and P.age >= V.age - 1

<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>19</td>
<td>98120</td>
<td>flu</td>
<td>p2</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>
Natural Join Example

<table>
<thead>
<tr>
<th>AnonPatient P</th>
<th>Voters V</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>age</strong></td>
<td><strong>zip</strong></td>
</tr>
<tr>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

\[ P \bowtie V \]
Natural Join

- Given schemas $R(A, B, C, D), S(A, C, E)$, what is the schema of $R \bowtie S$?

- Given $R(A, B, C), S(D, E)$, what is $R \bowtie S$?

- Given $R(A, B), S(A, B)$, what is $R \bowtie S$?
More Joins

• **Outer join**
  – Include tuples with no matches in the output
  – Use NULL values for missing attributes

• **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>

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 ⨝ 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>p1</td>
</tr>
<tr>
<td>20</td>
<td>98120</td>
<td>flu</td>
<td>p2</td>
</tr>
<tr>
<td>33</td>
<td>98120</td>
<td>lung</td>
<td>null</td>
</tr>
</tbody>
</table>
Example of Algebra Queries

Q1: Names of patients who have heart disease

\[ \pi_{\text{name}}(\text{Voter} \bowtie (\sigma_{\text{disease}=\text{‘heart’}} (\text{AnonPatient}))) \]
More Examples

Relations

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

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

(Many more examples in the R&G)
Logical Query Plans

An RA expression but represented as a tree

\[ \Pi_{\text{sname}} \]

\[ \sigma_{\text{psize} > 10} \]

\[ \sigma_{\text{psize} > 10} \]

\[ \Pi_{\text{sname}} \]

\[ \sigma_{\text{psize} > 10} \]

\[ \Pi_{\text{sname}} \]

\[ \sigma_{\text{psize} > 10} \]

\[ \Pi_{\text{sname}} \]

\[ \sigma_{\text{psize} > 10} \]

\[ \Pi_{\text{sname}} \]

\[ \sigma_{\text{psize} > 10} \]
More Joins

- Semi-join = the subset of $R$ that joins with $S$
  
  $$R \bowtie S = \Pi_{\text{Attr}(R)}(R \bowtie S)$$

- Anti-semi join = the subset of $R$ that doesn’t join with $S$
  
  $$R - (R \bowtie S)$$
Extended Operators of Relational Algebra

• **Duplicate elimination** ($\delta$)
  - Since commercial DBMSs operate on *multisets/bags* not sets

• **Grouping and aggregate operators** ($\gamma$)
  - Partitions tuples of a relation into “groups”
  - Aggregates can then be applied to groups
  - Min, max, sum, average, count

• **Sort operator** ($\tau$)
From SQL to RA

• Every SQL query can (and is) translated to RA
Translating SQL to RA

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

Answer

\[ \Pi_{\text{city, q}} \sigma_c > 100 \left( \begin{array}{c}
T1(	ext{city, q, c}) \\
\end{array} \right) \]

\[ \gamma_{\text{city, sum(quantity)\rightarrow q, count(*) \rightarrow c}} \]

T1, T2 = temporary tables

sales(product, city, quantity)

T1, T2 = temporary tables
How about Subqueries?

Find all supplies in Washington who sell only products ≤ $100
How about Subqueries?

Find all supplies in Washington who sell only products ≤ $100

```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?

Find all supplies in Washington who sell only products ≤ $100

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?

Find all supplies in Washington who sell only products ≤ $100

```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)
```

De-Correlation

```sql
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?

Find all supplies in Washington who sell only products ≤ $100

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

EXCEPT = set difference

Un-nesting
How about Subqueries?

Find all supplies in Washington who sell only products $\leq 100$

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

Finally…
Relational Calculus

RC = First Order Logic \( (\land, \lor, \neg, \forall, \exists) \)

A query is \{expr \mid FOL-predicate\}

Two variants

• Tuple relational calculus query; uses tuple variables
• Domain relational calculus

E.g. names of suppliers that sell only products > $100

\[
\{ s.\text{name} \mid s \in \text{Supplier} \land \forall p \ (p \in \text{Supply} \rightarrow p.\text{price} > 100) \}
\]

\[
\{ n \mid \exists s,c,t \ (\text{Supplier}(s,n,c,t) \land \forall p,q,p (\text{Supply}(s,p,q,pr) \rightarrow pr > 100)) \}
\]
Example

• Set division: $R(A,B)/S(B)$
  - Defined as the largest set $T(A)$ such that $T \times S \subseteq R$
  - Equivalently: the set of A’s s.t. they occur with all B’s
  - Example:
    Takes(student, courseName), Course(courseName)
    Takes/Course = the students who took all courses.

• In class, or at home:
  - Define set division in RC
  - Convert to RA