CSE 544
Principles of Database Management Systems

Lecture 2 – Relational Algebra and SQL

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

Product

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

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

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

"selection" and "projection"

Eliminating Duplicates

```
SELECT  DISTINCT category
FROM    Product
```

Category
Gadgets
Photography
Household

Compare to:

```
SELECT  category
FROM    Product
```

Category
Gadgets
Gadgets
Photography
Household

Ordering the Results

```
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

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

Joins

Product (<u>pname</u>, price, category, manufacturer)
Company (<u>cname</u>, 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

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

$$\texttt{SELECT } a_1, a_2, \ldots, a_k \quad \texttt{FROM } R_1 \text{ AS } x_1, R_2 \text{ AS } x_2, \ldots, R_n \text{ AS } x_n \quad \texttt{WHERE } \text{Conditions}$$

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 $\cup$ $\{(a_1,\ldots,a_k)\}$
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

3. Compute the SELECT clause:
   grouped attributes and aggregates.

1&2. FROM-WHERE-GROUPBY

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	~~0.5~~	~~50~~
Banana	2	10
Banana	4	10

WHERE price > 1

3. SELECT

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	~~0.5~~	~~50~~
Banana	2	10
Banana	4	10

Product	TotalSales
Bagel	40
Banana	20

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

```
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 al 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 $a_1,\ldots,a_k$ and/or any aggregates but NO OTHER ATTRIBUTES

C1 = is any condition on the attributes in $R_1,\ldots,R_n$

C2 = is any condition on aggregate expressions and on attributes $a_1,\ldots,a_k$

Semantics of SQL With Group-By

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

Evaluation steps:

1. Evaluate FROM-WHERE using Nested Loop Semantics
2. Group by the attributes $a_1,\ldots,a_k$
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)

Existential quantifiers

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)

Existential quantifiers

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

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

Existential quantifiers

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

Existential quantifiers

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

Find all companies that make some products with price < 200

Now let's unnest it:

```
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

Universal quantifiers

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. <u>some</u> product ≥ 200

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

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

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE  C.cid NOT IN (SELECT P.cid
                                     FROM Product P
                                     WHERE P.price >= 200)
```

Subqueries in WHERE

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

Universal quantifiers

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

Universal quantifiers

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: $\sigma_{condition}(S)$
- Projection: $\pi_{list\text{-}of\text{-}attributes}(S)$
- 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

no	name	zip	disease
1	p1	98125	flu
2	p2	98125	heart
3	p3	98120	lung
4	p4	98120	heart

$\pi_{zip,disease}$(Patient)

zip	disease
98125	flu
98125	heart
98120	lung
98120	heart

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

no	name	zip	disease
2	p2	98125	heart
4	p4	98120	heart

π_{zip} ($\sigma_{disease=\text{'heart'}}$(Patient))

zip
98120
98125

Cross-Product Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \times V$

P.age	P.zip	disease	name	V.age	V.zip
54	98125	heart	p1	54	98125
54	98125	heart	p2	20	98120
20	98120	flu	p1	54	98125
20	98120	flu	p2	20	98120

Join Galore

- **Theta-join**: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
  - Join of R and S with a join condition θ
  - Cross-product followed by selection θ
- **Equijoin**: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
  - Theta-join where θ 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

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120
p3	20	98123

$P \bowtie_{P.age=V.age} V$

P.age	P.zip	P.disease	V.name	V.age	V.zip
54	98125	heart	p1	54	98125
20	98120	flu	p2	20	98120
20	98120	flu	p3	20	98123

Theta-Join Example

AnonPatient P

age	zip	disease
50	98125	heart
19	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$$P \bowtie_{P.zip = V.zip \text{ and } P.age <= V.age + 1 \text{ and } P.age >= V.age - 1} V$$

P.age	P.zip	P.disease	V.name	V.age	V.zip
19	98120	flu	p2	20	98120

Natural Join Example

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

$P \bowtie V$

age	zip	disease	name
54	98125	heart	p1
20	98120	flu	p2

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

age	zip	disease
54	98125	heart
20	98120	flu
33	98120	lung

Voters V

name	age	zip
p1	54	98125
p2	20	98120

P ⟕ V

age	zip	disease	name
54	98125	heart	p1
20	98120	flu	p2
33	98120	lung	null

Example of Algebra Queries

Q1: Names of patients who have heart disease

$\pi_{name}(\text{Voter} \bowtie (\sigma_{disease=\text{'heart'}} (\text{AnonPatient}))$

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

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

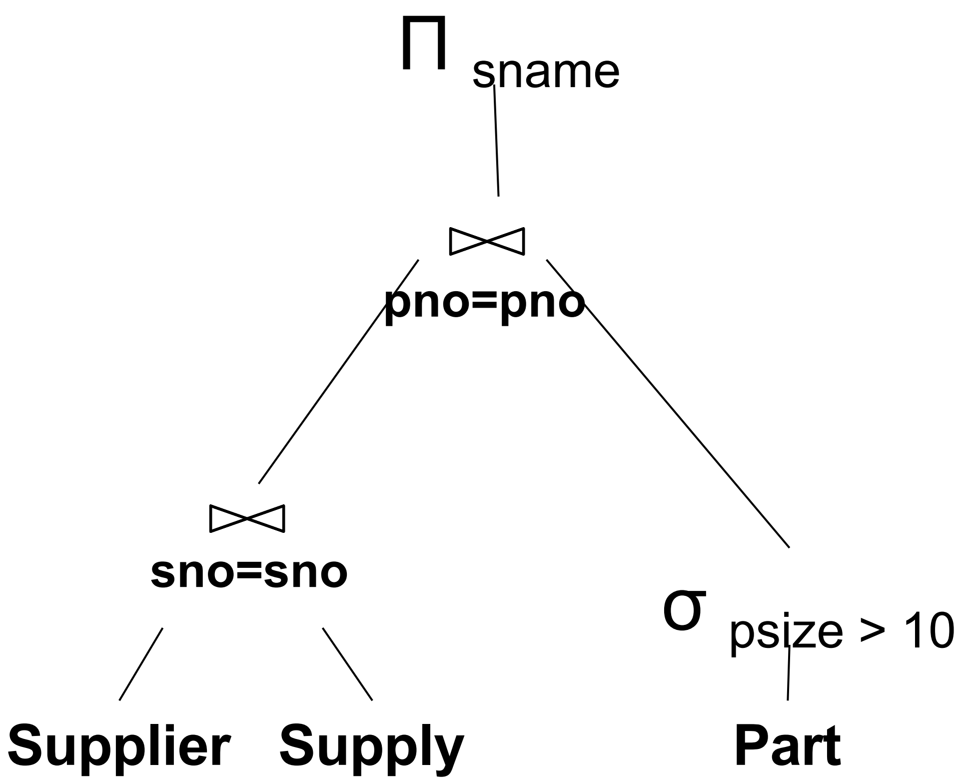
Q3: Name of supplier of red parts or parts with size greater than 10

$\pi_{sname}(\text{Supplier} \bowtie \text{Supply} \bowtie (\sigma_{psize>10}(\text{Part}) \cup \sigma_{pcolor='red'}(\text{Part})))$

(Many more examples in the R&G)

Logical Query Plans

An RA expression but represented as a tree



More Joins

- Semi-join = the subset of R that joins with S

  $R \ltimes S = \Pi_{Attr(R)}(R \bowtie S)$

- Anti-semi join = the subset of R that doesn't join with S

  $R - (R \ltimes S)$

Extended Operators of Relational Algebra

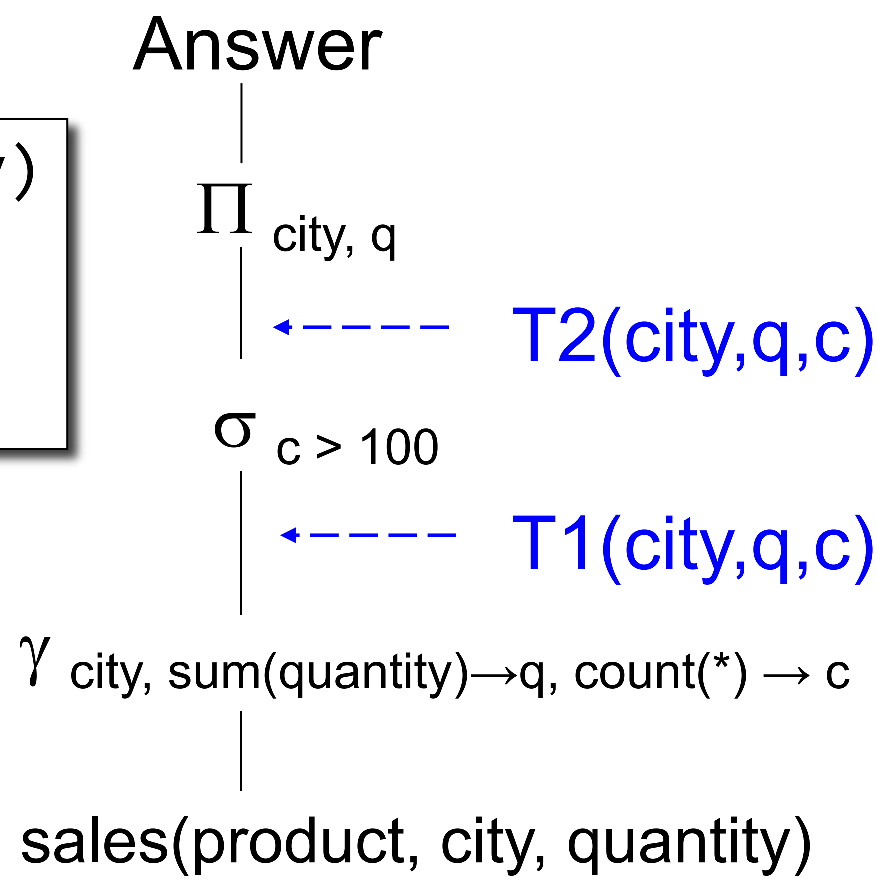
- Duplicate elimination (δ)
  – Since commercial DBMSs operate on **multisets/bags** not sets
- Grouping and aggregate operators (γ)
  – Partitions tuples of a relation into “groups”
  – Aggregates can then be applied to groups
  – Min, max, sum, average, count
- Sort operator (τ)

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
```



T1, T2 = temporary tables

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)

How about Subqueries?

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

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)

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)
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)

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)
```

Correlation !

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)

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)
```

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)
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)

How about Subqueries?

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

Un-nesting

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = 'WA')
    EXCEPT
 (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

```
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)
```

EXCEPT = set difference

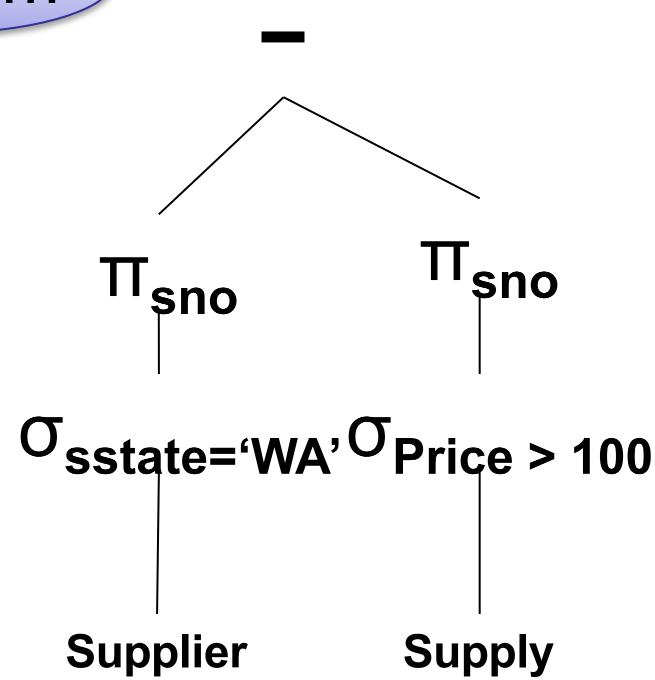
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)

How about Subqueries?

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

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = ‘WA’)
    EXCEPT
 (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

Finally…



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

Relational Calculus

RC = First Order Logic ($\wedge$, $\vee$, $\neg$, $\forall$, $\exists$)

A query is {expr | FOL-predicate}

Two variants

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

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

$\{ s.name \mid s \in Supplier \wedge \forall p\, (p \in Supply \rightarrow p.price > 100)\}$

$\{ n \mid \exists s,c,t\, (Supplier(s,n,c,t) \wedge \forall p,q,p (Supply(s,p,q,pr) \rightarrow pr > 100)\}$

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