CSE544
Data Management

Lecture 2

SQL and Relational Algebra

Announcements

- Thursday (tomorrow):
  - Makeup lecture at10:30 in CSE2 371
- Monday: no class (MLK day)
- Tuesday: project groups due
- Wednesday: first review due
- Next Saturday: homework 1 due

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

- 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

- 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/limiting the Results

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

Ascending, unless you specify the DESC keyword.

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.

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
```

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

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE 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

NULLs in SQL

- A NULL value means missing, or unknown, or undefined, or inapplicable
- We can specify whether attributes may or may not be NULL:

```
CREATE TABLE product
   (pid int NOT NULL,
    pname text NOT NULL,
    price int          – may be NULL
   );
```

Three-Valued Logic

- False=0, Unknown=0.4, True=1
- Result of a comparison A=B is
  - False or True when both A, B are not null
  - Unknown otherwise
- AND, OR, NOT are min, max, 1-.

Three-Valued Logic

- False=0, Unknown=0.4, True=1
- Result of a comparison A=B is
  - False or True when both A, B are not null
  - Unknown otherwise
- AND, OR, NOT are min, max, 1-.

```
select *
from Product
where (price <= 100) or (price > 100)
```

pid	Pname	price
1	iPhone	500
2	iPod	80
3	iPad	NULL

Three-Valued Logic

- False=0, Unknown=0.4, True=1
- Result of a comparison A=B is
  - False or True when both A, B are not null
  - Unknown otherwise
- AND, OR, NOT are min, max, 1-.

```
select *
from Product
where (price <= 100) or (price > 100)
```

```
where (price <= 100) or (price > 100)
      or isNull(price)
```

pid	Pname	price
1	iPhone	500
2	iPod	80
3	iPad	NULL

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

```
Product(name, category)
Purchase(prodName, store)

-- prodName is foreign key
```

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

```
Product(name, category)
Purchase(prodName, store)

-- prodName is foreign key
```

```
SELECT x.name, x.category, y.store
FROM   Product x, Purchase y
WHERE  x.name = y.prodName
```

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

```
Product(name, category)
Purchase(prodName, store)

-- prodName is foreign key
```

```
SELECT x.name, x.category, y.store
FROM   Product x, Purchase y
WHERE  x.name = y.prodName
```

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

```
Product(name, category)
Purchase(prodName, store)

-- prodName is foreign key
```

```
SELECT x.name, x.category, y.store
FROM   Product x, Purchase y
WHERE  x.name = y.prodName
```

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

missing

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Output

Name	Category	Store
Gizmo	gadget	Wiz
Camera	Photo	Ritz
Camera	Photo	Wiz

Outer joins

Retrieve all product names, categories, and stores where they were purchased. Include products that never sold

```
Product(name, category)
Purchase(prodName, store)

-- prodName is foreign key
```

```
SELECT x.name, x.category, y.store
FROM   Product x LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
```

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Output

Name	Category	Store
Gizmo	gadget	Wiz
Camera	Photo	Ritz
Camera	Photo	Wiz
OneClick	Photo	NULL

Joins

- **Inner join** = includes only matching tuples (i.e. regular join)
- **Left outer join** = includes everything from the left
- **Right outer join** = includes everything from the right
- **Full outer join** = includes everything

ON v.s. WHERE

- Outer join condition in the ON clause
- Different from the WHERE clause
- Compare:

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
   AND y.price=10
```

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE y.price=10
```

Aggregation

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

```
SELECT  count(*)
FROM    Product
WHERE   maker='Toyota'
```

Aggregation

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

```
SELECT  count(*)
FROM    Product
WHERE   maker='Toyota'
```

SQL supports several aggregation operations:
sum, count, min, max, avg

Aggregation

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

```
SELECT  count(*)
FROM    Product
WHERE   maker='Toyota'
```

SQL supports several aggregation operations:
sum, count, min, max, avg

Duplicates are kept unless DISTINCT
Nulls are "ignored"

Aggregation

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

```
SELECT  count(*)
FROM    Product
WHERE   maker='Toyota'
```

SQL supports several aggregation operations:
sum, count, min, max, avg

Duplicates are kept unless DISTINCT
Nulls are "ignored"

```
SELECT  count(*)
FROM    Product
WHERE   maker='Toyota'
```

```
SELECT  count(model)
FROM    Product
WHERE   maker='Toyota'
```

```
SELECT  count(DISTINCT model)
FROM    Product
WHERE   maker='Toyota'
```

Grouping and Aggregation

Purchase(date, product, price, quantity)

For each product, find the total quantity of sales over $1

Grouping and Aggregation

Purchase(date, product, price, quantity)

For each product, find the total quantity of sales over $1

```
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 GROUP BY
3. Compute the SELECT clause:
   grouped attributes and aggregates.

3. SELECT

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

WHERE price > 1

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

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

Returns 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 brought in revenue > $1000.

```
SELECT      product, Sum(quantity)
FROM        Purchase
WHERE       price > 1
GROUP BY    product
HAVING      Sum(price* quantity) > 1000
```

HAVING clause contains conditions on aggregates.

WHERE vs HAVING

WHERE condition is applied to individual rows

- Keep or drop the row
- No aggregates allowed in WHERE

HAVING condition is applied to the entire group

- Keep or drop the group
- May use aggregate functions in HAVING

Syntax & Semantics

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

Sytnax:

- R1, …, Rn = tables to be joined
- C1 = is any condition on the attributes in $R_1,\ldots,R_n$
- C2 = is any condition on aggregate expressions
- and on attributes $a_1,\ldots,a_k$
- S = may contain attributes $a_1,\ldots,a_k$ and/or any aggregates but NO OTHER ATTRIBUTES

Syntax & Semantics

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

Semantics

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 <u>some</u> products with price < 200

Using EXISTS:

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

Subqueries in WHERE

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

Existential quantifiers

Find all companies that make some products with price < 200

Now let's unnest it:

```
SELECT DISTINCT C.cid, 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)

Universal quantifiers

Find all companies that make <u>only</u> products with price < 200

same as:

Find all companies whose products <u>all</u> 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 C.cid, C.cname
FROM    Company C
WHERE  C.cid IN (SELECT P.cid
                    FROM Product P
                    WHERE P.price >= 200)
```

Subqueries in WHERE

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

```
SELECT C.cid, 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. all their products have price < 200

```
SELECT C.cid, 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 C.cid, 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)

Universal quantifiers

Find all companies that make only products with price < 200

Using ALL:

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

Monotone Queries

- **Definition**: A query Q is called monotone if:
  - Whenever we add a tuple to a table…
  - …we do not lose any tuple from the output

Monotone Queries

- **Definition**: A query Q is called monotone if:
  - Whenever we add a tuple to a table…
  - …we do not lose any tuple from the output
- SELECT * FROM R -- is moontone
  SELECT count(*) FROM R -- is not

Monotone Queries

- **Definition**: A query Q is called monotone if:
  - Whenever we add a tuple to a table…
  - …we do not lose any tuple from the output
- SELECT * FROM R -- is moontone
  SELECT count(*) FROM R -- is not
- **Fact**: All queries without subqueries or aggregates are monotone.
  Proof: nested loop semantics

Monotone Queries

- **Definition**: A query Q is called monotone if:
  - Whenever we add a tuple to a table…
  - …we do not lose any tuple from the output
- SELECT * FROM R -- is moontone
  SELECT count(*) FROM R -- is not
- **Fact**: All queries without subqueries or aggregates are monotone.
  Proof: nested loop semantics
- **Fact** "Find all companies that make <u>only</u> products with price < 200" is not monotone (proof in class)

Monotone Queries

- **Definition**: A query Q is called monotone if:
  - Whenever we add a tuple to a table…
  - …we do not lose any tuple from the output
- SELECT * FROM R -- is moontone
  SELECT count(*) FROM R -- is not
- **Fact**: All queries without subqueries or aggregates are monotone.
  Proof: nested loop semantics
- **Fact** “Find all companies that make <u>only</u> products with price < 200” is not monotone (proof in class)
- Hence, it cannot be flattened without aggregates

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

Set v.s. Bag Semantics

- Sets: {a,b,d,e}; {1,7,8,12,19}
- Bags: {a,a,b}, {1,7,7,2,2,2,8,9,9}
- SQL bag semantics
- Relational Algebra: either set semantics or bag semantics

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 $\rho(R(F),E)$

Selection & Projection

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

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

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

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

zip
98120
98125

Cross-Product

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

AnonPatient × Voters

P.age	P.zip	P.disease	V.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

Many Types of Joins

- **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

AnonPatient P $\bowtie_{P.age=V.age}$ Voters 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?

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

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

  $R - (R \ltimes S)$

Example of Algebra Queries

Q1: Names of patients who have heart disease

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

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

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

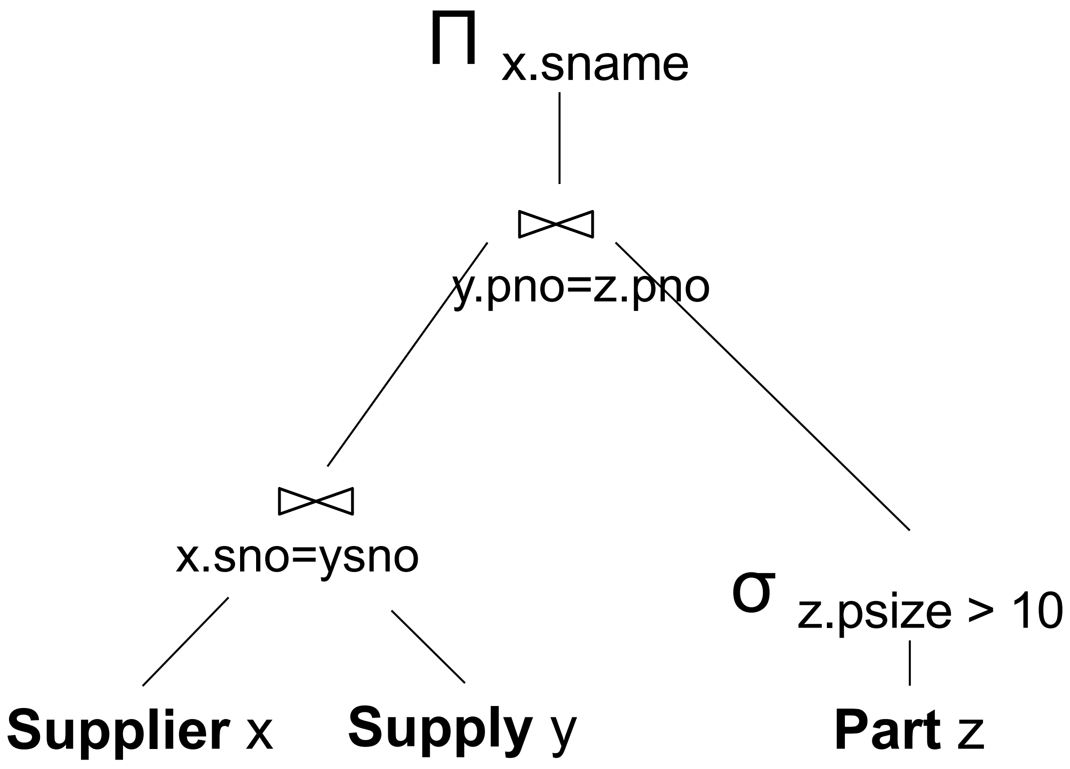
Q3: Name of supplier of red parts or parts with size greater than 10

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

(Many more examples in the R&G)

Logical Query Plans

An RA expression but represented as a tree



Extended Operators of Relational Algebra

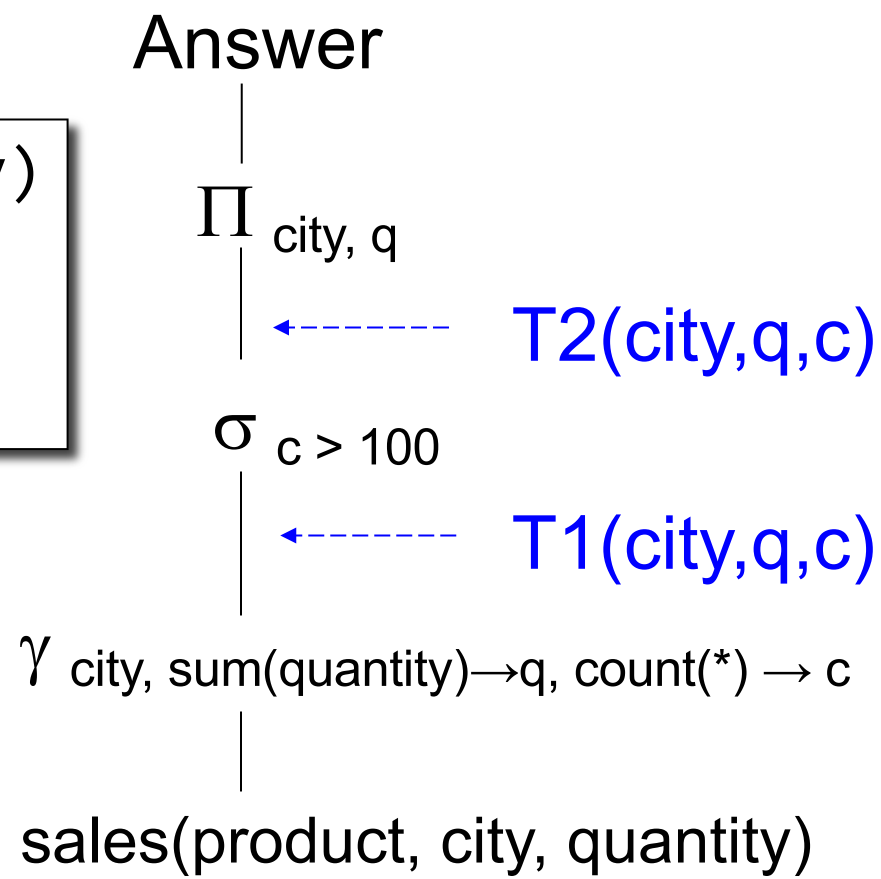
- Duplicate elimination (δ)
  - Since commercial DBMSs operate on **multisets/bags** not sets
- Grouping and aggregate operators (γ)
  - Partitions tuples of a relation into "groups"
  - Aggregates can then be applied to groups
  - Min, max, sum, average, count
- Sort operator (τ)

From SQL to RA

- Every SQL query can (and is) translated to RA

Translating SQL to RA

```sql
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

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)
```

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

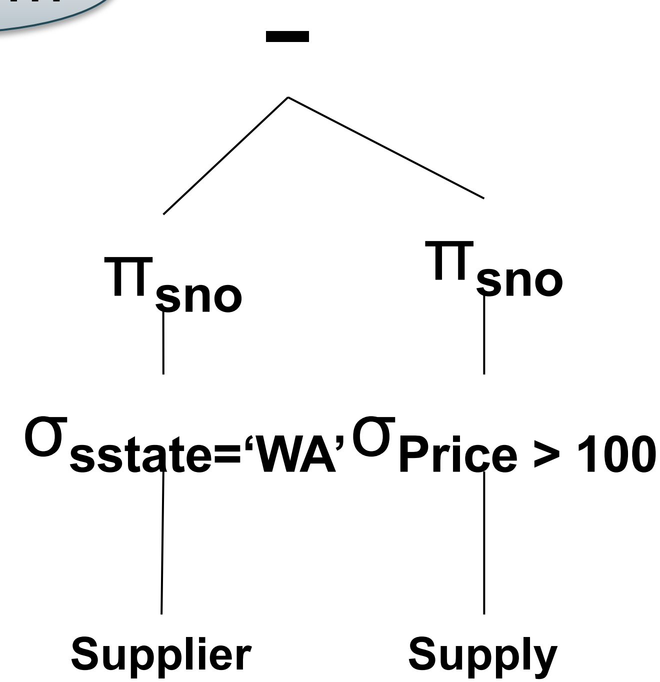
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,price)

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 *<u>all</u>* 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