CSE 344

JUNE 29TH

RELATIONAL ALGEBRA

ADMINISTRIVIA

- **WQ2 due tonight**
- **HW2 out (Due Wednesday)**
  - git pull upstream master
- **Canvas page published**
  - only grades there… all else on usual web site

REVIEW

- **SQL: complete query language**
  - inner & outer joins (FROM & WHERE clauses)
  - group by: replaces rows with groups of rows
    - from then on, only group-by columns and aggregation
  - having filter on groups (vs where on rows)
  - order by
  - select is processed **last**
  - **subqueries** can appear in from (no worries), select, where
    - in select, result must be 1 row, 1 column (i.e. single value)
      - in principle, should unnest
    - in where, can be single value or use EXISTS, IN, ANY/ALL
      - in principle, existential (exists, in, any) should unnest
- **Where we left off: can all queries be unnested?**

QUESTION FOR DATABASE THEORY FANS AND THEIR FRIENDS

Can we unnest *universal quantifier* query?

We first need the concept of <u>monotonicity</u>

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

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

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

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

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c004
Camera	149.99	c003

Company

cid	cname	city
c002	Sunworks	Bonn
c001	DB Inc.	Lyon
c003	Builder	Lodtz

$\xrightarrow{Q}$

pname	city
Gizmo	Lyon
Camera	Lodtz

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

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

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c004
Camera	149.99	c003

Company

cid	cname	city
c002	Sunworks	Bonn
c001	DB Inc.	Lyon
c003	Builder	Lodtz

Q

pname	city
Gizmo	Lyon
Camera	Lodtz

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c004
Camera	149.99	c003
iPad	499.99	c001

Company

cid	cname	city
c002	Sunworks	Bonn
c001	DB Inc.	Lyon
c003	Builder	Lodtz

Q

pname	city
Gizmo	Lyon
Camera	Lodtz
iPad	Lyon

So far it looks monotone...

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

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

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c004
Camera	149.99	c003

Company

cid	cname	city
c002	Sunworks	Bonn
c001	DB Inc.	Lyon
c003	Builder	Lodtz

Q

pname	city
Gizmo	Lyon
Camera	Lodtz

Product

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c004
Camera	149.99	c003
iPad	499.99	c001

Company

cid	cname	city
c002	Sunworks	Bonn
c001	DB Inc.	Lyon
c003	Builder	Lodtz
c004	Crafter	Lodtz

Q

Q is not monotone!

pname	city
Gizmo	Lodtz
Camera	Lodtz
iPad	Lyon

MONOTONE QUERIES

<u>Theorem</u>: If Q is a SELECT-FROM-WHERE query (i.e., Q has no subqueries and no aggregation), then Q is monotone.

MONOTONE QUERIES

Theorem: If Q is a SELECT-FROM-WHERE query (i.e., Q has no subqueries and no aggregation), then Q 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

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

```
for x₁ in R₁ do
  for x₂ in R₂ do
    …
      for xₙ in Rₙ do
        if Conditions
          output (a₁,…,aₖ)
```

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

MONOTONE QUERIES

The query:

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

is not monotone

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

MONOTONE QUERIES

The query:

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

is not monotone

pname	price	cid
Gizmo	19.99	c001

cid	cname	city
c001	Sunworks	Bonn

⇒

cname
Sunworks

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

MONOTONE QUERIES

The query:

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

is not monotone

pname	price	cid
Gizmo	19.99	c001

cid	cname	city
c001	Sunworks	Bonn

cname
Sunworks

pname	price	cid
Gizmo	19.99	c001
Gadget	999.99	c001

cid	cname	city
c001	Sunworks	Bonn

cname

Consequence: If a query is not monotonic, then we cannot write it as a SELECT-FROM-WHERE query without grouping or nested subqueries

Purchase(pid, product, quantity, price)

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 ?

```
Author(login,name)
Wrote(login,url)
```

MORE UNNESTING

Find authors who wrote ≥ 10 documents:

```
Author(login,name)
Wrote(login,url)
```

MORE UNNESTING

Find authors who wrote ≥ 10 documents:

Attempt 1: with nested queries

This is SQL by a novice

```
SELECT DISTINCT Author.name
FROM       Author
WHERE       (SELECT count(Wrote.url)
             FROM Wrote
             WHERE Author.login=Wrote.login)
               >= 10
```

```
Author(login,name)
Wrote(login,url)
```

MORE UNNESTING

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
```

This is
SQL by
an expert

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

FINDING WITNESSES

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

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

FINDING WITNESSES

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

Finding the maximum price is easy…

```sql
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

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

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;
```

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

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;
```

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

FINDING WITNESSES

Or we can use a subquery in where clause

```sql
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);
```

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

FINDING WITNESSES

There is a more concise solution here:

```sql
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)
```

Why do we need v.price here?
(It doesn't change groups!)

REVIEW

- **SQL: complete query language**
  - inner & outer joins
  - group by: replaces rows with groups of rows
    - from then on, only group-by columns and aggregation
  - having filter on groups (vs where on rows)
  - order by
  - select is processed **last**
  - subqueries
- **SQL is used everywhere for relational data**

RELATIONAL ALGEBRA

- **Remember from last week**
  - SQL queries are combinations of functions on tables
    - joins: (R1, R2, …, Rk) ~> R
    - where: R ~> R’
  - Each one receives tables as input and has a table as an output

RELATIONAL ALGEBRA

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

In SQL we say *<u>what</u>* we want **("declarative")**

In RA we can express *<u>how</u>* to get it **("imperative")**

An RA expression is also called a *<u>query plan</u>*

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

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: σ, π, δ, etc

SETS V.S. BAGS

Sets: {a,b,c}, {a,d,e,f}, { }, . . .

Bags: {a, a, b, c}, {b, b, b, b, b}, . . .

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 σ

Projection π

Cartesian product X**, join** $\bowtie$

(Rename ρ**)**

Duplicate elimination δ

Grouping and aggregation γ

Sorting τ

RA

Extended RA

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

UNION AND DIFFERENCE

$$R1 \cup R2$$
$$R1 - R2$$

Only make sense if R1, R2 have the same schema

What do they mean over bags ?

WHAT ABOUT INTERSECTION ?

Derived operator using minus

$$R1 \cap R2 = R1 - (R1 - R2)$$

Derived using join

$$R1 \cap R2 = R1 \bowtie R2$$

SELECTION

Returns all tuples which satisfy a condition

$$\boxed{\sigma_c(R)}$$

Examples

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

The condition c can be =, <, <=, >, >=, <> combined with AND, OR, NOT

Employee

SSN	Name	Salary
1234545	John	20000
5423341	Smith	60000
4352342	Fred	50000

$\sigma_{Salary > 40000}$ (Employee)

SSN	Name	Salary
5423341	Smith	60000
4352342	Fred	50000

PROJECTION

Eliminates columns

$$\boxed{\pi_{A1,\ldots,An}(R)}$$

Example: project social-security number and names:

- $\pi_{SSN,\ Name}$ (Employee) → Answer(SSN, Name)

Different semantics over sets or bags! Why?

Employee

SSN	Name	Salary
1234545	John	20000
5423341	John	60000
4352342	John	20000

$\pi_{Name,Salary}$ (Employee)

Name	Salary
John	20000
John	60000
John	20000

Bag semantics

Name	Salary
John	20000
John	60000

Set semantics

Which is more efficient?

COMPOSING RA OPERATORS

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

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

zip	disease
98125	heart
98120	heart

CARTESIAN PRODUCT

Each tuple in R1 with each tuple in R2

$$\boxed{R1 \times R2}$$

Rare in practice; mainly used to express joins

CROSS-PRODUCT EXAMPLE

Employee

Name	SSN
John	999999999
Tony	777777777

Dependent

EmpSSN	DepName
999999999	Emily
777777777	Joe

Employee X Dependent

Name	SSN	EmpSSN	DepName
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

NATURAL JOIN

$$\boxed{R1 \bowtie R2}$$

Meaning: $\mathbf{R1 \bowtie R2 = \Pi_A(\sigma_\theta (R1 \times R2))}$

Where:

- Selection σ_θ checks equality of all common attributes (i.e., attributes with same names)
- Projection Π_A eliminates duplicate common attributes

NATURAL JOIN EXAMPLE

R

A	B
X	Y
X	Z
Y	Z
Z	V

S

B	C
Z	U
V	W
Z	V

R ⋈ S =
$\Pi_{ABC}(\sigma_{R.B=S.B}(R \times S))$

A	B	C
X	Z	U
X	Z	V
Y	Z	U
Y	Z	V
Z	V	W

NATURAL JOIN EXAMPLE 2

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
Alice	54	98125
Bob	20	98120

$P \bowtie V$

age	zip	disease	name
54	98125	heart	Alice
20	98120	flu	Bob

AnonPatient (age, zip, disease)
Voters (name, age, zip)

THETA JOIN

A join that involves a predicate

$$R1 \bowtie_{\theta} R2 \quad = \quad \sigma_{\theta}\,(R1 \times R2)$$

Here θ can be any condition

No projection in this case!

For our voters/patients example:

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

EQUIJOIN

A theta join where θ is an equality predicate

$$\boxed{R1 \bowtie_{\theta} R2 \quad = \sigma_{\theta}\,(R1 \times R2)}$$

By far the most used variant of join in practice

What is the relationship with natural join?

EQUIJOIN EXAMPLE

AnonPatient P

age	zip	disease
54	98125	heart
20	98120	flu

Voters V

name	age	zip
p1	54	98125
p2	20	98120

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

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

JOIN SUMMARY

Theta-join: $R \bowtie_\theta S = \sigma_\theta (R \times S)$

- Join of R and S with a join condition θ
- Cartesian product followed by selection θ
- No projection

Equijoin: $R \bowtie_\theta S = \sigma_\theta (R \times S)$

- Join condition θ 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 π_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