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 monotonicity
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

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

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pname</strong></td>
<td><strong>cid</strong></td>
</tr>
<tr>
<td>Gizmo</td>
<td>c001</td>
</tr>
<tr>
<td>Gadget</td>
<td>c004</td>
</tr>
<tr>
<td>Camera</td>
<td>c003</td>
</tr>
</tbody>
</table>

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

So far it looks monotone...
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 (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.
The query:

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

is not monotone
MONOTONE QUERIES

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>
MONOTONE QUERIES

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 grouping or nested subqueries.
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 ≥ 10 documents:
Find authors who wrote ≥ 10 documents:

**Attempt 1: with nested queries**

```
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
```

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
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
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;
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;
```
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:

```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!)
• **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 *what* we want ("declarative")
In RA we can express *how* to get it ("imperative")

An RA expression is also called a *query plan*

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: $\sigma$, $\pi$, $\delta$, 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 $\sigma$
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 $R_1$, $R_2$ 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

\[ \sigma_c(R) \]

Examples

- \( \sigma_{\text{Salary} > 40000} \) (Employee)
- \( \sigma_{\text{name} = \text{"Smith"}} \) (Employee)

The condition c can be =, <, <=, >, >=, <> 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>

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

<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

Set semantics

Which is more efficient?
COMPOSING RA OPERATORS

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>

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

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

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

Each tuple in R1 with each tuple in R2

\[ R1 \times R2 \]

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

## Employee

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
</tr>
</tbody>
</table>

## Dependent

<table>
<thead>
<tr>
<th>EmpSSN</th>
<th>DepName</th>
</tr>
</thead>
<tbody>
<tr>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>777777777</td>
<td>Joe</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>999999999</td>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>John</td>
<td>999999999</td>
<td>777777777</td>
<td>Joe</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
<td>777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>
NATURAL JOIN

Meaning:  \( R_1 \bowtie R_2 = \Pi_A(\sigma_\theta(R_1 \times R_2)) \)

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

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

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

#### EXAMPLE 2

**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>Alice</td>
<td>54</td>
<td>98125</td>
</tr>
<tr>
<td>Bob</td>
<td>20</td>
<td>98120</td>
</tr>
</tbody>
</table>

$P \bowtie V$
A join that involves a predicate

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

Here \( \theta \) can be any condition
No projection in this case!
For our voters/patients example:

\[ P \bowtie P.zip = V.zip \text{ and } P.age \geq V.age - 1 \text{ and } P.age \leq V.age + 1 \]
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 \join_{P.age=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 \( \theta \)
- Cartesian 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.