CSE544
Data Management
Lecture 2
SQL and Relational Algebra
Announcements

• Thursday (tomorrow):
  – Makeup lecture at 10:30 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)
  - ...

CSE 544 - Winter 2020
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

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

```
<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
```

“selection” and “projection”
Eliminating Duplicates

```
SELECT DISTINCT category
FROM Product
```

Compare to:

```
SELECT category
FROM Product
```

<table>
<thead>
<tr>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gadgets</td>
</tr>
<tr>
<td>Photography</td>
</tr>
<tr>
<td>Household</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gadgets</td>
</tr>
<tr>
<td>Gadgets</td>
</tr>
<tr>
<td>Photography</td>
</tr>
<tr>
<td>Household</td>
</tr>
</tbody>
</table>

CSE 544 - Winter 2020
Ordering/limiting the Results

```sql
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 (pname, price, category, manufacturer)
Company (cname, 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

```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

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-

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

<table>
<thead>
<tr>
<th>pid</th>
<th>Pname</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iPhone</td>
<td>500</td>
</tr>
<tr>
<td>2</td>
<td>iPod</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>iPad</td>
<td>NULL</td>
</tr>
</tbody>
</table>
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-

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

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

<table>
<thead>
<tr>
<th>pid</th>
<th>Pname</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iPhone</td>
<td>500</td>
</tr>
<tr>
<td>2</td>
<td>iPod</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>iPad</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Outer joins

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

-- prodName is foreign key

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

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

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

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
```

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

**Product**

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
</tbody>
</table>

**Purchase**

<table>
<thead>
<tr>
<th>ProdName</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>
Outer joins

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

-- prodName is foreign key

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

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

<table>
<thead>
<tr>
<th>Product</th>
<th>Purchase</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Category</td>
<td>ProdName</td>
</tr>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td>Gizmo</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>Camera</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
<td>Camera</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>Wiz</td>
</tr>
</tbody>
</table>
Outer joins

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

-- prodName is foreign key

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

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

| Product | | Purchase | | Output |
|---------|---------|----------|--------|
| Name    | Category| ProdName | Store  |
| Gizmo   | gadget  | Gizmo    | Wiz    |
| Camera  | Photo   | Camera   | Ritz   |
| OneClick| Photo   | Camera   | Wiz    |
|         |         |          |        |
|         |         |          |        |
|         |         |          |        |
|         |         |          |        |
|         |         |          |        |

Now it's present
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:

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

```sql
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

**Example SQL Queries:**

- `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

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

Duplicates are kept unless DISTINCT
Nulls are “ignored”
Grouping and Aggregation

Purchase(date, product, price, quantity)

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

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** clause.

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

<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

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

<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
```

Returns ONE TUPLE per group

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>40</td>
</tr>
<tr>
<td>Banana</td>
<td>20</td>
</tr>
</tbody>
</table>
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

Syntax:
- \( R_1, \ldots, R_n = \text{tables to be joined} \)
- \( C_1 = \text{is any condition on the attributes in } R_1, \ldots, R_n \)
- \( C_2 = \text{is any condition on aggregate expressions} \)
- \( \text{and on attributes } a_1, \ldots, a_k \)
- \( S = \text{may contain attributes } a_1, \ldots, a_k \text{ and/or any aggregates but NO OTHER ATTRIBUTES} \)
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 a₁,...,aₖ
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 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)

Find all companies that make some products with price < 200

Using IN

```sql
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)

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

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
```
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 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 ≥ 200

```sql
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

```sql
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)

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)

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 monotone
  **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 monotone
  **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 monotone
  **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 only 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 monotone
- **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 **only** 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**: $\bigcup$
- **Set difference**: $\neg$
- **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

<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>

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

<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>

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

<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>

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

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

## 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>

## AnonPatient × Voters

<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>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>
Many Types of Joins

- **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>

AnonPatient P \( \bowtie_{P.age = V.age} \) Voters 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>
<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 \bowtie 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

### 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 \bowtie 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>
</tbody>
</table>
Natural Join

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

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

• Given $R(A, B)$, $S(A, B)$, what is $R \Join S$?
### 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>
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 $S$

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

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

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

An RA expression but represented as a tree

\[
\begin{align*}
\Pi & \quad \text{x.sname} \\
\land & \quad \text{y.pno=z.pno} \\
\land & \quad \text{x.sno=ysno} \\
\text{Supplier x} & \quad \text{Supply y} \\
\text{Part z} & \quad \sigma \quad \text{z.psize > 10}
\end{align*}
\]
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

\[
\text{SELECT city, sum(quantity) FROM sales GROUP BY city HAVING count(*) > 100}
\]

\[
\begin{align*}
& \text{T1, T2 = temporary tables} \\
& \text{sales(product, city, quantity)}
\end{align*}
\]

Answer

\[
\begin{align*}
\Pi &_{\text{city, q}} \quad T2(\text{city,q,c}) \\
\sigma &_{c > 100} \quad T1(\text{city,q,c}) \\
\gamma &_{\text{city, sum(quantity)→q, count(*) → c}}
\end{align*}
\]
How about Subqueries?

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

```
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)
```

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

\[(\text{SELECT } Q\text{.sno} \text{ FROM Supplier } Q \text{ WHERE } Q\text{.sstate} = 'WA') \text{ EXCEPT } (\text{SELECT } P\text{.sno} \text{ FROM Supply } P \text{ WHERE } P\text{.price} > 100)\]  

EXCEPT = set difference
How about Subqueries?

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

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

\textit{RC} = \text{First Order Logic} (\land, \lor, \neg, \forall, \exists)

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

Two variants

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

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

\{ \text{s.name} \mid \text{s} \in \text{Supplier} \land \forall \text{p} (\text{p} \in \text{Supply} \rightarrow \text{p.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