Database Management Systems
CSEP 544

Lecture 2: SQL
Announcements

• HW1 due tonight (11:59pm)

• PA2 & HW2 released

• Fill out HW3 email account form by tonight!

• Final information posted on piazza

• Check website for up to date OH info
Review

• Data models
  – Instance
  – Schema
  – Language

• Relational data model
  – Relations are flat
  – Tuples are not ordered

• Logical and physical data independence
Reading Assignment 1
Selections in SQL

```
SELECT *  
FROM Product  
WHERE price > 100.0
```

Projections in SQL

```
SELECT CName  
FROM Product
```
Joins in SQL

Retrieve all Japanese products that cost < $150

<table>
<thead>
<tr>
<th>Product</th>
<th>price</th>
<th>category</th>
<th>manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiTouch</td>
<td>199.99</td>
<td>gadget</td>
<td>Canon</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>49.99</td>
<td>photography</td>
<td>Canon</td>
</tr>
<tr>
<td>Gizom</td>
<td>50</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SuperGizmo</td>
<td>250.00</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>Japan</td>
</tr>
</tbody>
</table>
Joins in SQL

Retrieve all Japanese products that cost < $150

```
SELECT pname, price
FROM Product, Company
WHERE ...
```
Joins in SQL

Retrieve all Japanese products that cost < $150

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>category</th>
<th>manufacturer</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiTouch</td>
<td>199.99</td>
<td>gadget</td>
<td>Canon</td>
<td>Japan</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>49.99</td>
<td>photography</td>
<td>Canon</td>
<td>Japan</td>
</tr>
<tr>
<td>Gizom</td>
<td>50</td>
<td>gadget</td>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
<tr>
<td>SuperGizmo</td>
<td>250.00</td>
<td>gadget</td>
<td>GizmoWorks</td>
<td>Japan</td>
</tr>
</tbody>
</table>

SELECT pname, price
FROM Product, Company
WHERE manufacturer=cname AND country='Japan' AND price < 150
Retrieve all Japanese products that cost < $150

```
SELECT P.pname, P.price
FROM Product as P, Company as C
WHERE P.manufacturer=C.cname AND 
    C.country='Japan' AND C.price < 150
```
Join in SQL

Product(pname, price, category, manufacturer)
Company(cname, country)

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>category</th>
<th>manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiTouch</td>
<td>199.99</td>
<td>gadget</td>
<td>Canon</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>49.99</td>
<td>photography</td>
<td>Canon</td>
</tr>
<tr>
<td>Gizom</td>
<td>50</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SuperGizmo</td>
<td>250.00</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cname</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>Japan</td>
</tr>
</tbody>
</table>

Retrieve all USA companies that manufacture “gadget” products
Joins in SQL

### Tables

**Product**
- **pname**
- **price**
- **category**
- **manufacturer**

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>category</th>
<th>manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>MultiTouch</td>
<td>199.99</td>
<td>gadget</td>
<td>Canon</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>49.99</td>
<td>photography</td>
<td>Canon</td>
</tr>
<tr>
<td>Gizom</td>
<td>50</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SuperGizmo</td>
<td>250.00</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
</tbody>
</table>

**Company**
- **cname**
- **country**

<table>
<thead>
<tr>
<th>cname</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>Japan</td>
</tr>
</tbody>
</table>

### Query

Retrieve all USA companies that manufacture “gadget” products

```sql
SELECT DISTINCT cname
FROM Product, Company
WHERE country='USA' AND category = 'gadget'
AND manufacturer = cname
```

**Why DISTINCT?**

The `DISTINCT` keyword is used to select only unique values from the `cname` column of the `Company` table. This is important because when joining the `Product` and `Company` tables, there might be repeated entries for the same company if the same company manufactures multiple products in different categories. The `DISTINCT` keyword ensures that each company name is listed only once in the result set.
Joins in SQL

• The standard join in SQL is sometimes called an inner join
  – Each row in the result must come from both tables in the join

• Sometimes we want to include rows from only one of the two table: outer join
Joins and Aggregates
(Inner) joins

Product(pname, price, category, manufacturer)
Company(cname, country)
-- manufacturer is foreign key to Company

SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
AND    manufacturer = cname
(Inner) joins

```
SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
       AND manufacturer = cname
```
(Inner) joins

```
SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
       AND manufacturer = cname
```
(Inner) joins

SELECT DISTINCT cname
FROM Product, Company
WHERE country='USA' AND category = 'gadget'
AND manufacturer = cname

<table>
<thead>
<tr>
<th>pname</th>
<th>category</th>
<th>manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>Hitachi</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cname</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>Japan</td>
</tr>
</tbody>
</table>

•Product

•Company
(Inner) joins

```
SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
       AND manufacturer = cname
```

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pname</strong></td>
<td><strong>c</strong></td>
</tr>
<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>

<table>
<thead>
<tr>
<th><strong>pname</strong></th>
<th><strong>category</strong></th>
<th><strong>manufacturer</strong></th>
<th><strong>cname</strong></th>
<th><strong>country</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td>GizmoWorks</td>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
</tbody>
</table>
(Inner) joins

\[
SELECT \text{DISTINCT} \ \text{cname} \\
\text{FROM} \ \text{Product, Company} \\
\text{WHERE} \ \text{country}='\text{USA}' \ \text{AND} \ \text{category} = '\text{gadget}' \ \text{AND} \ \text{manufacturer} = \text{cname}
\]

- **Product**
  - pname | category | manufacturer
  - Gizmo | gadget   | GizmoWorks
  - Camera | Photo    | Hitachi
  - OneClick | Photo   | Hitachi

- **Company**
  - cname | country
  - GizmoWorks | USA
  - Canon | Japan
  - Hitachi | Japan
(Inner) joins

```
SELECT DISTINCT cname
FROM   Product, Company
WHERE  country='USA' AND category = 'gadget'
AND manufacturer = cname
```
(Inner) joins

```
SELECT DISTINCT cname
FROM Product, Company
WHERE country='USA' AND category = 'gadget'
    AND manufacturer = cname
```

```
SELECT DISTINCT cname
FROM Product JOIN Company ON
    country = 'USA' AND category = 'gadget'
    AND manufacturer = cname
```
(Inner) Joins

```
for x1 in R1:
    for x2 in R2:
        ...
        for xm in Rm:
            if Cond(x1, x2...):
                output(x1.a1, x2.a2, ... xm.am)
```

This is called nested loop semantics since we are interpreting what a join means using a nested loop
Another example

Product\( (pname, \text{price}, \text{category}, \text{manufacturer}) \)
Company\( (cname, \text{country}) \)

-- manufacturer is foreign key to Company

Retrieve all Japanese companies that manufacture products in both ‘gadget’ and ‘photography’ categories
Another example

Product(pname, price, category, manufacturer)
Company(cname, country)
-- manufacturer is foreign key to Company

Retrieve all Japanese companies that manufacture products in both 'gadget' and 'photography' categories

SELECT DISTINCT cname
FROM Product P1, Product P2, Company
WHERE country = 'Japan' AND P1.category = 'gadget'
    AND P2.category = 'photography'
    AND P1.manufacturer = cname
    AND P2.manufacturer = cname;
Self-Joins and Tuple Variables

• Find all companies that manufacture both products in the ‘gadgets’ and ‘photo’ category

• Joining Product with Company is insufficient: need to join Product, with Product, and with Company

• When a relation occurs twice in the FROM clause we call it a self-join
  – in that case we must use tuple variables (why?)
Self-joins

```
SELECT DISTINCT z.cname
FROM Product x, Product y, Company z
WHERE z.country = 'USA'
  AND x.category = 'gadget'
  AND y.category = 'photo'
  AND x.manufacturer = z.cname
  AND y.manufacturer = z.cname;
```

<table>
<thead>
<tr>
<th>Product</th>
<th></th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pname</strong></td>
<td><strong>category</strong></td>
<td><strong>manufacturer</strong></td>
</tr>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>photo</td>
<td>Hitachi</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>Photo</td>
<td>GizmoWorks</td>
</tr>
</tbody>
</table>
Self-joins

SELECT DISTINCT z.cname
FROM    Product x, Product y, Company z
WHERE   z.country = 'USA'
        AND x.category = 'gadget'
        AND y.category = 'photo'
        AND x.manufacturer = z.cname
        AND y.manufacturer = z.cname;

<table>
<thead>
<tr>
<th>Product</th>
<th>Name</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gizmo</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td></td>
<td>SingleTouch</td>
<td>photo</td>
<td>Hitachi</td>
</tr>
<tr>
<td></td>
<td>MultiTouch</td>
<td>Photo</td>
<td>GizmoWorks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company</th>
<th>Name</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
<tr>
<td></td>
<td>Hitachi</td>
<td>Japan</td>
</tr>
</tbody>
</table>
Self-joins

```
SELECT DISTINCT z.cname
FROM Product x, Product y, Company z
WHERE z.country = 'USA'
    AND x.category = 'gadget'
    AND y.category = 'photo'
    AND x.manufacturer = z.cname
    AND y.manufacturer = z.cname;
```
Self-joins

```
SELECT DISTINCT z.cname
FROM Product x, Product y, Company z
WHERE z.country = 'USA'
AND x.category = 'gadget'
AND y.category = 'photo'
AND x.manufacturer = z.cname
AND y.manufacturer = z.cname;
```
Self-joins

```
SELECT DISTINCT z.cname
FROM Product x, Product y, Company z
WHERE z.country = 'USA'
    AND x.category = 'gadget'
    AND y.category = 'photo'
    AND x.manufacturer = z.cname
    AND y.manufacturer = z.cname;
```
Self-joins

```
SELECT DISTINCT z.cname
FROM Product x, Product y, Company z
WHERE z.country = 'USA'
AND x.category = 'gadget'
AND y.category = 'photo'
AND x.manufacturer = z.cname
AND y.manufacturer = z.cname;
```
Self-joins

```
SELECT DISTINCT z.cname
FROM Product x, Product y, Company z
WHERE z.country = 'USA'
  AND x.category = 'gadget'
  AND y.category = 'photo'
  AND x.manufacturer = z.cname
  AND y.manufacturer = z.cname;
```
Self-joins

```sql
SELECT DISTINCT z.cname
FROM Product x, Product y, Company z
WHERE z.country = 'USA'
    AND x.category = 'gadget'
    AND y.category = 'photo'
    AND x.manufacturer = z.cname
    AND y.manufacturer = z.cname;
```
Outer joins

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

-- prodName is foreign key

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

We want to include products that are never sold, but some are not listed! Why?
Outer joins

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

-- prodName is foreign key

```sql
SELECT Product.name, Purchase.store
FROM   Product  LEFT OUTER JOIN Purchase ON
        Product.name = Purchase.prodName
```
SELECT Product.name, Purchase.store
FROM Product JOIN Purchase ON Product.name = Purchase.prodName

<table>
<thead>
<tr>
<th>Product</th>
<th>Purchase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Category</td>
</tr>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
<tr>
<td>ProdName</td>
<td>Store</td>
</tr>
<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>
### 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>
SELECT Product.name, Purchase.store
FROM Product JOIN Purchase ON Product.name = Purchase.prodName

Output

<table>
<thead>
<tr>
<th>Name</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
</tbody>
</table>

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>
SELECT Product.name, Purchase.store
FROM Product JOIN Purchase ON Product.name = Purchase.prodName
SELECT `Product.name`, `Purchase.store` 
FROM `Product` JOIN `Purchase` ON `Product.name` = `Purchase.prodName` 

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>ProdName</th>
<th>Store</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td>Gizmo</td>
<td>Wiz</td>
<td></td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>Camera</td>
<td>Ritz</td>
<td></td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
<td>Camera</td>
<td>Wiz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Gizmo</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>Wiz</strong></td>
</tr>
</tbody>
</table>
```sql
SELECT Product.name, Purchase.store
FROM Product JOIN Purchase
ON Product.name = Purchase.prodName
```

**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>

**Output**

<table>
<thead>
<tr>
<th>Name</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
</tbody>
</table>
### 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>
</tbody>
</table>

### Output

<table>
<thead>
<tr>
<th>Name</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
</tbody>
</table>
### 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>

### Output

<table>
<thead>
<tr>
<th>Name</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>

```sql
SELECT Product.name, Purchase.store
FROM Product JOIN Purchase ON Product.name = Purchase.prodName
```
SELECT Product.name, Purchase.store
FROM Product JOIN Purchase ON
Product.name = Purchase.prodName

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>

Output

<table>
<thead>
<tr>
<th>Name</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>
```sql
SELECT Product.name, Purchase.store
FROM Product
LEFT OUTER JOIN Purchase
ON Product.name = Purchase.prodName
```
### SQL Query

```sql
SELECT Product.name, Purchase.store
FROM Product
LEFT OUTER JOIN Purchase
ON Product.name = Purchase.prodName
```

### Product Table

<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

<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>

### Output Table

<table>
<thead>
<tr>
<th>Name</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>
<tr>
<td>OneClick</td>
<td>NULL</td>
</tr>
</tbody>
</table>

CSEP 544 - Fall 2017
```
SELECT Product.name, Purchase.store
FROM Product
FULL OUTER JOIN Purchase
ON Product.name = Purchase.prodName
```
Outer Joins

\[
\text{tableA (LEFT/RIGHT/FULL) OUTER JOIN tableB ON p}
\]

- **Left outer join:**
  - Include tuples from `tableA` even if no match
- **Right outer join:**
  - Include tuples from `tableB` even if no match
- **Full outer join:**
  - Include tuples from both even if no match

- **In all cases:**
  - Patch tuples without matches using NULL
Comment about SQLite

• Cannot load NULL values such that they are actually loaded as null values

• So we need to use two steps:
  – Load null values using some type of special value
  – Update the special values to actual null values

```sql
update Purchase
set price = null
where price = ‘null’
```
Simple Aggregations

Five basic aggregate operations in SQL

```sql
select count(*) from Purchase
select sum(quantity) from Purchase
select avg(price) from Purchase
select max(quantity) from Purchase
select min(quantity) from Purchase
```

Except count, all aggregations apply to a single attribute
Aggregates and NULL Values

Null values are not used in aggregates

```sql
insert into Purchase
values(12, 'gadget', NULL, NULL, 'april')
```

Try the following at home:

```sql
select count(*) from Purchase
select count(quantity) from Purchase

select sum(quantity) from Purchase

select count(*)
from Purchase
where quantity is not null;
```
COUNT applies to duplicates, unless otherwise stated:

```sql
SELECT count(product) same as count(*) if no nulls
FROM Purchase
WHERE price > 4.99
```

We probably want:

```sql
SELECT count(DISTINCT product)
FROM Purchase
WHERE price > 4.99
```
More Examples

SELECT Sum(price * quantity) FROM Purchase

SELECT Sum(price * quantity) FROM Purchase WHERE product = 'bagel'

What do they mean?
Grouping and Query Evaluation
Grouping and Aggregation

Purchase(product, price, quantity)

Find total quantities for all sales over $1, by product.
Grouping and Aggregation

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

<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>

<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>
Other Examples

Compare these two queries:

```sql
SELECT product, count(*)
FROM Purchase
GROUP BY product
```

```sql
SELECT month, count(*)
FROM Purchase
GROUP BY month
```

```sql
SELECT product,
    sum(quantity) AS SumQuantity,
    max(price) AS MaxPrice
FROM Purchase
GROUP BY product
```

What does it mean?
Need to be Careful...

```
SELECT product, max(quantity)
FROM Purchase
GROUP BY product
```

```
SELECT product, quantity
FROM Purchase
GROUP BY product
```

<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>
**Need to be Careful…**

SELECT product, quantity FROM Purchase GROUP BY product

<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>

Everything in SELECT must be either a GROUP-BY attribute, or an aggregate.
Need to be Careful…

```sql
SELECT product, max(quantity)
FROM Purchase
GROUP BY product
```

```sql
SELECT product, quantity
FROM Purchase
GROUP BY product
```

sqlite is WRONG on this query.

Advanced DBMS (e.g. SQL Server) gives an error

<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>
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
```

How is this query processed?
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

**SQL Query:**

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

**Table:**

<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>

FWGS
### 3.4. Grouping, Select

#### SQL Query

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

#### Table

<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>

#### Result

<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>

**FWGS**
Ordering Results

```sql
SELECT product, sum(price*quantity) as rev
FROM Purchase
GROUP BY product
ORDER BY rev desc
```

Note: some SQL engines want you to say ORDER BY sum(price*quantity) desc
HAVING Clause

Same query as before, 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.
General form of Grouping and Aggregation

\[
\text{SELECT } S \\
\text{FROM } R_1, \ldots, R_n \\
\text{WHERE } C_1 \\
\text{GROUP BY } a_1, \ldots, a_k \\
\text{HAVING } C_2
\]

\( S \) = may contain attributes \( a_1, \ldots, a_k \) and/or any aggregates but NO OTHER ATTRIBUTES

\( C_1 \) = is any condition on the attributes in \( R_1, \ldots, R_n \)

\( C_2 \) = is any condition on aggregate expressions and on attributes \( a_1, \ldots, a_k \)
Semantics of SQL With Group-By

```
SELECT  S
FROM    R_1,...,R_n
WHERE   C1
GROUP BY a_1,...,a_k
HAVING  C2
```

Evaluation steps:
1. Evaluate FROM-WHERE using Nested Loop Semantics
2. Group by the attributes a_1,...,a_k
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result
Exercise

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”
Exercise

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
FROM Purchase
```
Exercise

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
FROM Purchase
GROUP BY month
```
Exercise

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
FROM Purchase
GROUP BY month
HAVING sum(quantity) < 10
```
Exercise

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
SELECT month, sum(price*quantity), sum(quantity) as TotalSold
FROM Purchase
GROUP BY month
HAVING sum(quantity) < 10
```
Exercise

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
SELECT month, sum(price*quantity), sum(quantity) as TotalSold
FROM Purchase
GROUP BY month
HAVING sum(quantity) < 10
ORDER BY sum(quantity)
```
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 at all
  – May use aggregate functions in the group
Mystery Query

What do they compute?

```
SELECT month, sum(quantity), max(price)
FROM Purchase
GROUP BY month
```

```
SELECT month, sum(quantity)
FROM Purchase
GROUP BY month
```

```
SELECT month
FROM Purchase
GROUP BY month
```
Mystery Query

What do they compute?

```
SELECT  month, sum(quantity), max(price)
FROM    Purchase
GROUP BY month
```

```
SELECT  month, sum(quantity)
FROM    Purchase
GROUP BY month
```

```
SELECT  month
FROM    Purchase
GROUP BY month
```

Lesson: DISTINCT is a special case of GROUP BY
Product($pid$, $pname$, manufacturer)
Purchase($id$, $product_id$, price, month)

**Aggregate + Join**

For each manufacturer, compute how many products with price > $100 they sold
Aggregate + Join

For each manufacturer, compute how many products with price > $100 they sold

Problem: price is in Purchase, manufacturer is in Product...
Product(pid,pname,manufacturer)
Purchase(id,product_id,price,month)

Aggregate + Join

For each manufacturer, compute how many products with price > $100 they sold

Problem: price is in Purchase, manufacturer is in Product...

-- step 1: think about their join
SELECT ...
FROM Product x, Purchase y
WHERE x.pid = y.product_id
and y.price > 100

<table>
<thead>
<tr>
<th>manufacturer</th>
<th>...</th>
<th>price</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hitachi</td>
<td></td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>Canon</td>
<td></td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>Hitachi</td>
<td></td>
<td>180</td>
<td></td>
</tr>
</tbody>
</table>
For each manufacturer, compute how many products with price > $100 they sold

Problem: price is in Purchase, manufacturer is in Product...

-- step 1: think about their join
SELECT ...
FROM Product x, Purchase y
WHERE x.pid = y.product_id
    and y.price > 100

-- step 2: do the group-by on the join
SELECT x.manufacturer, count(*)
FROM Product x, Purchase y
WHERE x.pid = y.product_id
    and y.price > 100
GROUP BY x.manufacturer

<table>
<thead>
<tr>
<th>manufacturer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hitachi</td>
<td>150</td>
</tr>
<tr>
<td>Canon</td>
<td>300</td>
</tr>
<tr>
<td>Hitachi</td>
<td>180</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>manufacturer</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hitachi</td>
<td>2</td>
</tr>
<tr>
<td>Canon</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Aggregate + Join

Variant:
For each manufacturer, compute how many products with price > $100 they sold in each month

```
SELECT x.manufacturer, y.month, count(*)
FROM Product x, Purchase y
WHERE x.pid = y.product_id
   and y.price > 100
GROUP BY x.manufacturer, y.month
```

<table>
<thead>
<tr>
<th>manufacturer</th>
<th>month</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hitachi</td>
<td>Jan</td>
<td>2</td>
</tr>
<tr>
<td>Hitachi</td>
<td>Feb</td>
<td>1</td>
</tr>
<tr>
<td>Canon</td>
<td>Jan</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Including Empty Groups

• In the result of a group by query, there is one row per group in the result

```
SELECT x.manufacturer, count(*)
FROM Product x, Purchase y
WHERE x.pname = y.product
GROUP BY x.manufacturer
```

Count(*) is never 0
Including Empty Groups

```sql
SELECT x.manufacturer, count(y.pid)
FROM Product x
LEFT OUTER JOIN Purchase y
ON x.pname = y.product
GROUP BY x.manufacturer
```

Count(pid) is 0 when all pid’s in the group are NULL
Nested Queries
What have we learned so far

• Data models
• Relational data model
  – Instance: relations
  – Schema: table with attribute names
  – Language: SQL
What have we learned so far

• SQL features
  – Projections
  – Selections
  – Joins (inner and outer)
  – Aggregates
  – Group by
    – Inserts, updates, and deletes

• Make sure you read the textbook!
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
  - But sometimes it’s impossible, as we will see
Subqueries…

• Can appear as computed values in a SELECT clause

• Can appear in FROM clauses and aliased using a tuple variable that represents the tuples in the result of the subquery

• Can return a single constant to be compared with another value in a WHERE clause

• Can return relations to be used in WHERE clauses
1. Subqueries in SELECT

Product \((\text{pname}, \ \text{price}, \ \text{cid})\)
Company \((\text{cid}, \ \text{cname}, \ \text{city})\)

For each product return the city where it is manufactured

```
SELECT X.pname, (SELECT Y.city FROM Company Y WHERE Y.cid=X.cid) as City
FROM Product X
```

What happens if the subquery returns more than one city?
We get a runtime error
(and SQLite simply ignores the extra values…)

“correlated subquery”
Whenever possible, don’t use nested queries:

```
SELECT X.pname, (SELECT Y.city FROM Company Y WHERE Y.cid=X.cid) as City
FROM Product X
```

We have “unnested” the query
1. Subqueries in SELECT

Compute the number of products made by each company

```
SELECT DISTINCT C.cname, (SELECT count(*) FROM Product P WHERE P.cid=C.cid)
FROM Company C
```
1. Subqueries in SELECT

Compute the number of products made by each company

```sql
SELECT DISTINCT C.cname, (SELECT count(*) FROM Product P WHERE P.cid=C.cid)
FROM Company C
```

Better: we can unnest using a GROUP BY

```sql
SELECT C.cname, count(*)
FROM Company C, Product P
WHERE C.cid=P.cid
GROUP BY C.cname
```
But are these really equivalent?

SELECT DISTINCT C.cname, (SELECT count(*)
    FROM Product P
    WHERE P.cid=C.cid)
FROM Company C

SELECT C.cname, count(*)
FROM Company C, Product P
WHERE C.cid=P.cid
GROUP BY C.cname
1. Subqueries in SELECT

But are these really equivalent?

```
SELECT DISTINCT C.cname, (SELECT count(*)
                      FROM Product P
                      WHERE P.cid=C.cid)
FROM Company C
```

```
SELECT C.cname, count(*)
FROM Company C, Product P
WHERE C.cid=P.cid
GROUP BY C.cname
```

```
SELECT C.cname, count(pname)
FROM Company C LEFT OUTER JOIN Product P
ON C.cid=P.cid
GROUP BY C.cname
```

No! Different results if a company has no products
2. Subqueries in FROM

Find all products whose prices is > 20 and < 500

```
SELECT X.pname
FROM (SELECT *
      FROM Product AS Y
      WHERE price > 20) as X
WHERE X.price < 500
```

Side note: This is not a correlated subquery. (why?)

Try unnest this query!
2. Subqueries in FROM

At the end of the lecture we will see that sometimes we really need a subquery and one option will be to put it in the FROM clause.
Product (pname, price, cid)
Company (cid, cname, city)

3. Subqueries in WHERE

Find all companies that make some products with price < 200
Product (pname, price, cid)
Company (cid, cname, city)

3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers
3. Subqueries in WHERE

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)
```
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers

Using **IN**

```sql
SELECT DISTINCT C.cname 
FROM Company C 
WHERE C.cid IN (SELECT P.cid 
    FROM Product P 
    WHERE P.price < 200)
```
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Using \textbf{ANY}:

\[
\text{SELECT DISTINCT} \quad \text{C.cname} \\
\text{FROM} \quad \text{Company C} \\
\text{WHERE} \quad 200 > \text{ANY} \quad (\text{SELECT} \quad \text{price} \\
\text{FROM} \quad \text{Product P} \\
\text{WHERE} \quad \text{P.cid} = \text{C.cid})
\]
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Using **ANY**:

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

Existential quantifiers are not supported in SQLite.
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Now let’s unnest it:

```sql
SELECT DISTINCT C.cname
FROM Company C, Product P
WHERE C.cid = P.cid and P.price < 200
```
3. Subqueries in WHERE

Find all companies that make some products with price < 200

Existential quantifiers are easy!

Now let’s unnest it:

```
SELECT DISTINCT  C.cname
FROM    Company C, Product P
WHERE   C.cid = P.cid and P.price < 200
```
Product (pname, price, cid)
Company (cid, cname, city)

3. Subqueries in WHERE

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

same as:

Find all companies that make only products with price < 200
3. Subqueries in WHERE

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

same as:

Find all companies that make only products with price < 200
Product (pname, price, cid)
Company (cid, cname, city)

3. Subqueries in WHERE

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

same as:

Find all companies that make only products with price < 200

Universal quantifiers are hard! 😞
3. Subqueries in WHERE

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

1. Find the other companies that make some product ≥ 200

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

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

1. Find the other companies that make some product ≥ 200

```sql
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. all their products have price < 200

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

3. Subqueries in WHERE

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

Using EXISTS:

```sql
SELECT DISTINCT C.cname
FROM Company C
WHERE NOT EXISTS (SELECT *
  FROM Product P
  WHERE P.cid = C.cid AND P.price >= 200)
```
3. Subqueries in WHERE

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

Using **ALL**:

```
SELECT DISTINCT C.cname
FROM Company C
WHERE 200 >= ALL (SELECT price
                     FROM Product P
                     WHERE P.cid = C.cid)
```
3. Subqueries in WHERE

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

Using **ALL**:

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

Not supported in sqlite
Question for Database Theory
Fans and their Friends

• Can we unnest the universal quantifier query?

• We need to first discuss 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>price</strong></td>
</tr>
<tr>
<td>Gizmo</td>
<td>19.99</td>
</tr>
<tr>
<td>Gadget</td>
<td>999.99</td>
</tr>
<tr>
<td>Camera</td>
<td>149.99</td>
</tr>
</tbody>
</table>

Q
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

• **Theorem**: If Q is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it is monotone.
Monotone Queries

• **Theorem:** If $Q$ is a SELECT-FROM-WHERE query that does not have subqueries, and no aggregates, then it 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.

```sql
SELECT $a_1, a_2, \ldots, a_k$
FROM $R_1 AS x_1, R_2 AS x_2, \ldots, R_n AS x_n$
WHERE Conditions
```
Monotone Queries

• 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 nested subqueries
Queries that must be nested

- Queries with universal quantifiers or with negation

- Queries that use aggregates in certain ways
  - `sum(..)` and `count(*)` are NOT monotone, because they do not satisfy set containment
  - `select count(*) from R` is not monotone!