DATA 514

Lecture 2: Join and Aggregates in SQL
Announcements

• Did you remember WQ1 yesterday?
• HW1 is due tonight!

• WQ2 is out – due next Monday
• HW2 is out – due next Tuesdays
Outline

- Inner joins (6.2)
- Outer joins (6.3.8)
- Aggregations (6.4.3 – 6.4.6)
- Subqueries (6.3)
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>
</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>

Product(pname, price, category, manufacturer)
Company(cname, country)
Joins in SQL

<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>

Retrieve all Japanese products that cost < $150

```
SELECT  pname, price
FROM     Product, Company
WHERE    ...
```
Retrieve all Japanese products that cost < $150

```
SELECT  pname, price  
FROM    Product, Company  
WHERE   manufacturer=cname AND country='Japan' AND price < 150
```
Joins in SQL

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

Retrieve all USA companies that manufacture “gadget” products.

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

Retrieve all USA companies that manufacture "gadget" products

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

Why DISTINCT?
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**
(Inner) Joins

SELECT x1.a1, x2.a2, ... xm.am
FROM R1 as x1, R2 as x2, ... Rm as xm
WHERE Cond

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

```
SELECT x1.a1, x2.a2, ... xm.am 
FROM R1 as x1, R2 as x2, ... Rm as xm 
WHERE Cond
```

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

Nested loop semantics
(Inner) joins

Company\texttt{(cname, country)}
Product\texttt{(pname, price, category, manufacturer)}
   – manufacturer is foreign key

\begin{footnotesize}
\begin{verbatim}
SELECT DISTINCT cname
FROM    Product, Company
WHERE   country = 'USA' AND category = 'gadget' AND manufacturer = cname
\end{verbatim}
\end{footnotesize}
(Inner) joins

```sql
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>
(Inner) joins

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

<table>
<thead>
<tr>
<th><code>pname</code></th>
<th><code>category</code></th>
<th><code>manufacturer</code></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><code>cname</code></th>
<th><code>country</code></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
```
(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
```
(Inner) joins

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

```sql
SELECT DISTINCT cname
FROM Product JOIN Company
ON country = 'USA' AND category = 'gadget' AND manufacturer = cname
```
Self-Joins and Tuple Variables

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

• Just joining Product with Company is insufficient: instead need to join Product, with Product, 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?)
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>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<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>

<table>
<thead>
<tr>
<th>Company</th>
<th>Cname</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td></td>
<td>USA</td>
</tr>
<tr>
<td>Hitachi</td>
<td></td>
<td>Japan</td>
</tr>
</tbody>
</table>
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;
```

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>Hitachi</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>GizmoWorks</td>
</tr>
</tbody>
</table>
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>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
</tr>
<tr>
<td>Gizmo</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>photo</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>Photo</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

DATA 514- Winter 2018
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;
```

<table>
<thead>
<tr>
<th>Product</th>
<th>Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>Hitachi</td>
</tr>
<tr>
<td>MultiTouch</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>category</th>
<th>manufacturer</th>
</tr>
</thead>
<tbody>
<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>

<table>
<thead>
<tr>
<th>Company</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
<tr>
<td>Hitachi</td>
<td>Japan</td>
</tr>
</tbody>
</table>
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;

Product

<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>SingleTouch</td>
<td>photo</td>
<td>Hitachi</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>Photo</td>
<td>GizmoWorks</td>
</tr>
</tbody>
</table>

Company

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

Self-joins
### 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;
```

---

#### Product

<table>
<thead>
<tr>
<th>x.pname</th>
<th>x.category</th>
<th>x.manufacturer</th>
</tr>
</thead>
<tbody>
<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>

#### Company

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

---

#### Table

<table>
<thead>
<tr>
<th>x.pname</th>
<th>x.category</th>
<th>x.manufacturer</th>
<th>y.pname</th>
<th>y.category</th>
<th>y.manufacturer</th>
<th>z.cname</th>
<th>z.country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td>GizmoWorks</td>
<td>MultiTouch</td>
<td>Photo</td>
<td>GizmoWorks</td>
<td>GizmoWorks</td>
<td>USA</td>
</tr>
</tbody>
</table>
### 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;
```

#### Product

<table>
<thead>
<tr>
<th>x.pname</th>
<th>x.category</th>
<th>x.manufacturer</th>
<th>y.pname</th>
<th>y.category</th>
<th>y.manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td>GizmoWorks</td>
<td>MultiTouch</td>
<td>Photo</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>photo</td>
<td>Hitachi</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Company

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

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>Photo</td>
<td>28 USA</td>
</tr>
</tbody>
</table>
Outer joins

Product(name, category)
Purchase(prodName, store) -- prodName is foreign key

An “inner join”:

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

Same as:

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

But some Products are not listed! Why?
Outer joins

Product(name, category)  
Purchase(prodName, store) -- prodName is foreign key

If we want to include products that never sold, then we need an “outerjoin”:

```sql
SELECT Product.name, Purchase.store
FROM Product LEFT OUTER JOIN Purchase
ON Product.name = Purchase.prodName
```
SELECT Product.name, Purchase.store
FROM Product
LEFT OUTER 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>
</tbody>
</table>
### SELECT

```sql
SELECT Product.name, Purchase.store
FROM Product
LEFT OUTER 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>
SELECT Product.name, Purchase.store 
FROM Product \text{ LEFT OUTER JOIN} Purchase \text{ ON } 
Product.name = Purchase.prodName

<table>
<thead>
<tr>
<th>Product</th>
<th></th>
<th>Purchase</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>

| Product | | Store |
|---------|---------|
| Name    | Store   |
| Gizmo   | Wiz     |
SELECT Product.name, Purchase.store
FROM Product
LEFT OUTER JOIN Purchase
ON Product.name = Purchase.prodName

DATA 514- Winter 2018
```
SELECT Product.name, Purchase.store
FROM Product
LEFT OUTER 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>

**Name**

<table>
<thead>
<tr>
<th>Name</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
</tbody>
</table>
```
SELECT Product.name, Purchase.store
FROM Product LEFT OUTER JOIN Purchase ON Product.name = Purchase.prodName
```

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

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

DATA 514- Winter 2018
### Product

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

### SQL Query

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

DATA 514- Winter 2018
```sql
SELECT Product.name, Purchase.store
FROM Product
LEFT OUTER 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>

DATA 514- Winter 2018
```sql
SELECT Product.name, Purchase.store
FROM Product
LEFT OUTER 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>

---

DATA 514 - Winter 2018
```sql
SELECT Product.name, Purchase.store
FROM Product
LEFT OUTER JOIN Purchase
ON Product.name = Purchase.prodName
```

<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>

<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>

<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>

DATA 514- Winter 2018
### 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>

### SQL Query

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

• Left outer join:
  – Include the left tuple even if there’s no match

• Right outer join:
  – Include the right tuple even if there’s no match

• Full outer join:
  – Include both left and right tuples even if there’s no match
Aggregation in SQL

> sqlite3 lec02

sqlite> create table Purchase
        (pid int primary key,
         product text,
         price float,
         quantity int,
         month varchar(15));

sqlite> -- download lec02-data.txt
sqlite> .import lec02-data.txt Purchase
Comment about SQLite

• One 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

\[
\text{update Purchase}
\]
\[
\text{set price} = \text{null}
\]
\[
\text{where price} = \text{‘null’}
\]
Simple Aggregations

Five basic aggregate operations in 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')
```

Let’s try the following

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

select sum(quantity) from Purchase;

select sum(quantity)
from Purchase
where quantity is not null;
```
Aggregates and NULL Values

Null values are not used in aggregates

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

Let’s try the following

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

select sum(quantity) from Purchase;

select sum(quantity)
from Purchase
where quantity is not null;
```
Counting Duplicates

COUNT applies to duplicates, unless otherwise stated:

```
SELECT Count(product)
FROM Purchase
WHERE price > 4.99
```

same as Count(*) if no nulls

We probably want:

```
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?
Simple Aggregations

Purchase

<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>

\[
\text{SELECT } \text{Sum}(\text{price} \times \text{quantity}) \text{ FROM Purchase WHERE product = 'Bagel'}
\]

\[90 \text{ (= 60+30)}\]
### Simple Aggregations

#### Purchase

<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>

#### SQL Query

```sql
SELECT Sum(price * quantity) 
FROM Purchase 
WHERE product = 'Bagel'
```

90 (= 60+30)
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
```

Let’s see what this means…
Logical Query Processing
Phases

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-GROUPBY

<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

FWGS
### 3. SELECT

The SQL query is as follows:

```sql
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>

The total sales for each product are calculated as follows:

- **Bagel**: 3 * 20 = 60
- **Banana**: (0.5 * 50) + (2 * 10) + (4 * 10) = 20 + 20 + 40 = 80

**Total**: 40 + 20 = 60

**FWGS**
Other Examples

Compare these two queries:

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

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

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

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>
Ordering Results

```
SELECT Product, sum(price*quantity) as rev
FROM Purchase
GROUP BY Product
ORDER BY rev desc
```
Grouping and Aggregation

```
SELECT product, sum(price*quantity) as rev
FROM Purchase
GROUP BY product
```
SELECT product, sum(price*quantity) as rev
FROM Purchase
GROUP BY product
Ordering Results

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

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

FWGOS
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)
HAVING Clause

Same query as earlier, 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.
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”

```
SELECT month, sum(price*quantity),
       sum(quantity) 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”

\[
\text{SELECT month, sum(price*quantity), sum(quantity) as TotalSold}
\]

\text{FROM Purchase}

Purchase(pid, product, price, quantity, 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”

```
SELECT  month, sum(price*quantity),
        sum(quantity) 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”

```
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
create table Product
    (pid int primary key,
     pname varchar(15),
     manufacturer varchar(15));

insert into product values(1,'bagel','Sunshine Co.');
insert into product values(2,'banana','BusyHands');
insert into product values(3,'gizmo','GizmoWorks');
insert into product values(4,'gadget','BusyHands');
insert into product values(5,'powerGizmo','PowerWorks');
Aggregate + Join Example

What do these queries mean?

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

SELECT x.manufacturer, y.month, count(*)
FROM Product x, Purchase y
WHERE x.pname = y.product
GROUP BY x.manufacturer, y.month
Nested Loop Semantics for SFW

```
SELECT x1.a1, x2.a2, ... xm.am
FROM R1 as x1, R2 as x2, ... Rm as xm
WHERE Cond
```

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

loop semantics
Nested Loop Semantics for SFW

\[
\text{SELECT } \ x1.a1, \ x2.a2, \ldots \ \text{xm.am} \\
\text{FROM } \ R1 \text{ as } x1, \ R2 \text{ as } x2, \ldots \ Rm \text{ as } xm \\
\text{WHERE } \ Cond
\]

for \(x1\) in \(R1\):
    for \(x2\) in \(R2\):
        ...
        for \(xm\) in \(Rm\):
            if \(\text{Cond}(x1, \ x2\ldots)\):
                output(x1.a1, x2.a2, \ldots \ \text{xm.am})

Nested loop semantics
Semantics for SFWGH

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

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

C1 = is any condition on the attributes in \(R_1, \ldots, R_n\)

C2 = is any condition on aggregate expressions and on attributes \(a_1, \ldots, a_k\)
Semantics for SFWGH

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
Semantics for SFWGH

SELECT S
FROM R_1, ..., R_n
WHERE C_1
GROUP BY a_1, ..., a_k
HAVING C_2

Evaluation steps:
1. Evaluate FROM-WHERE using Nested Loop Semantics
2. Group by the attributes a_1, ..., a_k
3. Apply condition C_2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

Execution order: FWGHOS
Empty Groups

• In the result of a group by query, there is one row per group in the result
• No group can be empty!
• In particular, \text{count}(*) is never 0

\begin{verbatim}
SELECT x.manufacturer, count(*)
FROM Product x, Purchase y
WHERE x.pname = y.product
GROUP BY x.manufacturer
\end{verbatim}
Empty Group Solution: Outer Join

\[
\text{SELECT } x.\text{manufacturer}, \text{ count}(y.\text{pid}) \\
\text{FROM Product } x \text{ LEFT OUTER JOIN } \text{ Purchase } y \\
\text{ON } x.\text{pname} = y.\text{product} \\
\text{GROUP BY } x.\text{manufacturer}
\]
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…

• Can return a single constant and this constant can be compared with another value in a WHERE clause
• Can return relations that can be used in various ways in WHERE clauses
• Can appear in FROM clauses, followed by a tuple variable that represents the tuples in the result of the subquery
• Can appear as computed values in a SELECT clause
1. Subqueries in SELECT

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

For each product return the city where it is manufactured
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
```
1. Subqueries in SELECT

Product (pname, price, cid)  
Company(cid, cname, 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
(SQLite simply ignores the extra values)
1. Subqueries in SELECT

Product (pname, price, cid)
Company(cid, cname, 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
(SQLite simply ignores the extra values)
1. Subqueries in SELECT

Product (pname, price, cid)
Company(cid, cname, 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
(SQLite simply ignores the extra values)
1. Subqueries in SELECT

Whenever possible, don’t use a nested queries:

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

II

```sql
SELECT X.pname, Y.city
FROM Product X, Company Y
WHERE X.cid=Y.cid
```
1. Subqueries in SELECT

Whenever possible, don’t use a nested query:

```
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

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

Better: we can unnest by using a GROUP BY

```
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
```
1. Subqueries in SELECT

But are these really equivalent?

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

No! Different results if a company has no products

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

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

Find all products whose prices is > 20 and < 500

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

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.

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

- Some SQL engines (including sqlite!) support the WITH statement, which is equivalent to (and nicer than) subqueries in the FROM clause:

```
SELECT max(X.c) FROM (SELECT count(*) as c FROM company GROUP BY city) as X WITH Temp AS (SELECT count(*) as c FROM company GROUP BY city) SELECT max(X.c) FROM Temp
```
3. Subqueries in WHERE

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

3. Subqueries in WHERE

Find all companies that make some products with price < 200
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

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 ANY:

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

Find all companies that make some products with price < 200

Using ANY:

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

Existential quantifiers

Not supported in sqlite
3. Subqueries in WHERE

Find all companies that make some products with price < 200

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 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
```

Existential quantifiers are easy! 😊
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
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)

Universal quantifiers are hard ! 😞
3. Subqueries in WHERE

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

1. Find the other companies: i.e. s.t. 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: i.e. s.t. 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)
```
3. Subqueries in WHERE

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

Using \textbf{EXISTS}:

\begin{verbatim}
SELECT DISTINCT C.cname
FROM Company C
WHERE NOT EXISTS (SELECT *
  FROM Product P
  WHERE P.cid = C.cid \textbf{and} P.price >= 200)
\end{verbatim}
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 Fans and their Friends

• Can we unnest the *universal quantifier* query?
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

---

**Product**

<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>
<tr>
<td>Gadget</td>
<td>999.99</td>
<td>c004</td>
</tr>
<tr>
<td>Camera</td>
<td>149.99</td>
<td>c003</td>
</tr>
</tbody>
</table>

**Company**

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>c002</td>
<td>Sunworks</td>
<td>Bonn</td>
</tr>
<tr>
<td>c001</td>
<td>DB Inc.</td>
<td>Lyon</td>
</tr>
<tr>
<td>c003</td>
<td>Builder</td>
<td>Lodtz</td>
</tr>
</tbody>
</table>

**Q**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Lyon</td>
</tr>
<tr>
<td>Camera</td>
<td>Lodtz</td>
</tr>
</tbody>
</table>
**Monotone Queries**

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 (pname, price, cid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
</tr>
<tr>
<td>Gadget</td>
</tr>
<tr>
<td>Camera</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company (cid, cname, city)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c002</td>
</tr>
<tr>
<td>c001</td>
</tr>
<tr>
<td>c003</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Lyon</td>
</tr>
<tr>
<td>Camera</td>
<td>Lodtz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product (pname, price, cid)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
</tr>
<tr>
<td>Gadget</td>
</tr>
<tr>
<td>Camera</td>
</tr>
<tr>
<td>iPad</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Company (cid, cname, city)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c002</td>
</tr>
<tr>
<td>c001</td>
</tr>
<tr>
<td>c003</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Lyon</td>
</tr>
<tr>
<td>Camera</td>
<td>Lodtz</td>
</tr>
<tr>
<td>iPad</td>
<td>Lyon</td>
</tr>
</tbody>
</table>
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.
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:

\[
\text{Find all companies s.t. all their products have price < 200 is not monotone}
\]

\[
\begin{array}{ccc}
\text{pname} & \text{price} & \text{cid} \\
\hline
\text{Gizmo} & 19.99 & \text{c001} \\
\text{Gadget} & 999.99 & \text{c001} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{cid} & \text{cname} & \text{city} \\
\hline
\text{c001} & \text{Sunworks} & \text{Bonn} \\
\end{array}
\]

- Consequence: 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 must be nested

• Queries with universal quantifiers or with negation

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