Announcements

• HW2 is out
  – due next Tuesday 11pm
  – same format as HW1
  – uses joins, aggregation, grouping

• WQ2 due Sunday 11pm
Outline

• Last time:
  – outer joins
  – how to aggregate over all rows

• Grouping & aggregations (6.4.3 – 6.4.6)
Aggregation

Purchase(product, price, quantity)

Find number of bagels sold for more than $1

```
SELECT Sum(quantity) as TotalSold
FROM Purchase
WHERE price > 1 and product = 'bagel'
```
Grouping and Aggregation

Purchase(product, price, quantity)

Find number sold for more than $1 for each product

SELECT product, Sum(quantity)
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
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

```
SELECT product, SUM(quantity)
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>sum(quantity)</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:

```
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
```

How about this one?
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.

Better 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) 
FROM Purchase 
GROUP BY product 
ORDER BY sum(price*quantity) DESC

FWGOS
Ordering Results

```
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.

```sql
SELECT    product, sum(price*quantity)
FROM       Purchase
WHERE      price > 1
GROUP BY   product
HAVING     sum(quantity) > 30
```

HAVING clause contains conditions on groups.
Exercise

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

```sql
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
```

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');
Let’s figure out what these mean…

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

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

Purchase(pid, product, price, quantity, month)
Product(pid, pname, manufacturer)
Nested Loop Semantics for SFW

```sql
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)
Semantics for SFWGH

\[
\begin{align*}
\text{SELECT} & \quad S \\
\text{FROM} & \quad R_1,\ldots,R_n \\
\text{WHERE} & \quad C_1 \\
\text{GROUP BY} & \quad a_1,\ldots,a_k \\
\text{HAVING} & \quad C_2
\end{align*}
\]

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 \(a_1,\ldots,a_k\)

Why ?
Semantics for SFWGH

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

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

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

What do these queries mean?

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

```
SELECT manufacturer, month, count(*)
FROM Product, Purchase
WHERE pname = product
GROUP BY manufacturer, month
```
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, $\text{count}(\ast)$ is never 0

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

What if there are no purchases for a manufacturer
Empty Group Solution: Outer Join

```
SELECT manufacturer, count(quantity)
FROM Product
LEFT OUTER JOIN Purchase
ON pname = product
GROUP BY manufacturer
```

Why not `count(*)`?
Exercise:

Find all manufacturers with more than 10 items sold. Return manufacturer name and number of items sold.

```
SELECT manufacturer, sum(quantity)
FROM Product, Purchase
WHERE pname = product
GROUP BY manufacturer
HAVING sum(quantity) > 10
```
Exercise:

Find all manufacturers with more than 1 distinct product sold
Return the name of the manufacturer and number of distinct products sold

```
SELECT manufacturer, count(distinct product)
FROM Product, Purchase
WHERE pname = product
GROUP BY manufacturer
HAVING count(distinct product) > 1
```
Exercise:

Find all products with more than 2 purchases.
Return the name of the product and max price it was sold.

```
SELECT pname, max(price)
FROM Product, Purchase
WHERE pname = product
GROUP BY pname
HAVING COUNT(*) > 2
```
Exercise:

Find all manufacturers with at least 5 purchases in one month
Return manufacturer name, month, and number of items sold

```
SELECT manufacturer, month, sum(quantity)
FROM Product, Purchase
GROUP BY manufacturer, month
HAVING count(*) > 2
```