Announcements

- HW1 is due next Monday, 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

```sql
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

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

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.

Let's see what this means...
### 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

### 3. SELECT

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Bagel</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>0.5</td>
<td>50</td>
</tr>
<tr>
<td>Banana</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

- SELECT product, sum(quantity) AS SumQuantity
- SELECT product, max(quantity) AS MaxPrice
- SELECT month, count(*) FROM Purchase GROUP BY month
- SELECT product, quantity FROM Purchase GROUP BY product

#### Other Examples

- `SELECT count(*) AS Count FROM Purchase GROUP BY product`
- `SELECT count(*) FROM Purchase GROUP BY month`

#### Need to be Careful…

- `SELECT product, sum(quantity) FROM Purchase GROUP BY product`
- `SELECT product, sum(quantity) AS rev FROM Purchase GROUP BY product ORDER BY rev DESC`

SQLite allows this query to be executed with strange behavior. Better DBMS (e.g., SQL Server) gives an error.

### Ordering Results

- `SELECT product, sum(price*quantity) FROM Purchase GROUP BY product ORDER BY sum(price*quantity) DESC`
- `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 groups.

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

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)
```

Mystery Query

What do they compute?

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

Lesson:
DISTINCT is a special case of GROUP BY

Aggregates and Joins

```
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

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

Let’s figure out what these mean…

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

\[
\begin{align*}
&\text{SELECT } x_1.a_1, x_2.a_2, \ldots, x_m.a_m \\
&\text{FROM } R_1 \text{ as } x_1, R_2 \text{ as } x_2, \ldots, R_m \text{ as } x_m \\
&\text{WHERE } \text{Cond}
\end{align*}
\]

for \( x_1 \) in \( R_1 \):
for \( x_2 \) in \( R_2 \):
\[ \vdots \]
for \( x_m \) in \( R_m \):
if \( \text{Cond}(x_1, x_2, \ldots) \):
output(\( x_1.a_1, x_2.a_2, \ldots, x_m.a_m \))

Semantics for SFWGH

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

\( S \) = 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 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 \( C_2 \) to each group (may have aggregates)
4. Compute aggregates in \( S \) and return the result

Aggregate + Join Example

\[
\begin{align*}
&\text{SELECT } \text{manufacturer}, \text{count(*)} \\
&\text{FROM } \text{Product}, \text{Purchase} \\
&\text{WHERE } \text{pname} = \text{product} \\
&\text{GROUP BY } \text{manufacturer}
\end{align*}
\]

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

\[
\begin{align*}
&\text{SELECT } \text{manufacturer}, \text{count(*)} \\
&\text{FROM } \text{Product}, \text{Purchase} \\
&\text{WHERE } \text{pname} = \text{product} \\
&\text{GROUP BY } \text{manufacturer}
\end{align*}
\]
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:

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

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

Purchase(pid, product, price, quantity, month)
Product(pid, pname, manufacturer)

Exercise:

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

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

Product(pid, pname, manufacturer)
Purchase(pid, product, price, quantity, month)

Exercise:

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

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

Product(pid, pname, manufacturer)
Purchase(pid, product, price, quantity, month)

Exercise:

```
SELECT manufacturer, month, sum(quantity)
FROM Product, Purchase
WHERE pname = product
GROUP BY manufacturer, month
HAVING count(*) >= 5
```

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

Product(pid, pname, manufacturer)
Purchase(pid, product, price, quantity, month)