Introduction to Data Management
CSE 344

Lecture 5: Grouping and Query Evaluation
Announcements

• Web quiz 2 is open: due Tuesday 11pm

• Homework 2 is released: Wednesday 11pm
  – TA office hours
Review

• Selection
• Projection
• Join
  – Inner and outer
• Aggregates
Today

• Aggregations and grouping (6.4.3 – 6.4.6)
• Order of query evaluation
Grouping and Aggregation

Purchase(product, price, quantity)

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

### 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>

### 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>
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…

\[
\text{SELECT product, max(quantity) from Purchase group by product}
\]

\[
\text{SELECT product, quantity from Purchase group by product}
\]

<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 FROM Purchase GROUP BY product

- SELECT quantity FROM Purchase

- 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>

Can’t project a non-grouped / non-aggregated column!
Need to be Careful…

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

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

SQLite is WRONG on this query.

```
<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>
```

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

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>
3.4. Grouping, Select

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

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

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 \(a_1, \ldots, a_k\)
Semantics of SQL With Group-By

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

SELECT $S$
FROM $R_1, \ldots, R_n$
WHERE $C_1$
GROUP BY $a_1, \ldots, a_k$
HAVING $C_2$
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”
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”

```sql
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?

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

What do they compute?

\[
\text{SELECT month, sum(quantity), max(price)} \\
\text{FROM Purchase} \\
\text{GROUP BY month}
\]

\[
\text{SELECT month, sum(quantity)} \\
\text{FROM Purchase} \\
\text{GROUP BY month}
\]

\[
\text{SELECT month} \\
\text{FROM Purchase} \\
\text{GROUP BY month}
\]

Lesson: DISTINCT is a special case of GROUP BY
Aggregate + Join Example

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

<table>
<thead>
<tr>
<th>manufacturer</th>
<th>month</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>canon</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>canon</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>sony</td>
<td>4</td>
<td>50</td>
</tr>
</tbody>
</table>
Empty Groups

- In the result of a group by query, there is one row per group in the result.
- No group can be empty!
  - i.e., count(*) is never 0

```
SELECT x.manufacturer, count(*)
FROM Product x, Purchase y
WHERE x.pname = y.product
GROUP BY x.manufacturer
```
Empty Group Solution: Outer Join

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