Database Systems
CSE 344

Lectures 5: Grouping & Aggregation
Wednesday June 28
Announcements

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

• WQ2 due Sunday 11pm
Outline

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

• Today:
  – Grouping & Aggregations (6.4.3 – 6.4.6)
Aggregation

Purchase(product, price, quantity)

Find number of bagels sold for more than $1

\[
\text{SELECT } \text{Sum(quantity) as Sold}
\text{FROM } \text{Purchase}
\text{WHERE price > 1 and product = 'bagel'}
\]
Grouping and Aggregation

Purchase(product, price, quantity)

Find number sold for more than $1 for each product

Group By

New Keyword
Grouping and Aggregation

```
SELECT product, Sum(quantity) AS Sales
FROM Purchase
WHERE price > 1
GROUP BY product
```
Grouping and Aggregation

Purchase(product, price, quantity)

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

```
SELECT product, Sum(quantity) AS Sales
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 **GROUP BY**

3. Compute the **SELECT** clause: grouped attributes and aggregates.

FWGS

Execution Order
SELECT product, Sum(quantity) AS Sales 
FROM Purchase 
WHERE price > 1 
GROUP BY product
### 3.4. Grouping, Select

#### SQL Query

```sql
SELECT product, Sum(quantity) AS Sales
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>

<table>
<thead>
<tr>
<th>Product</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>40</td>
</tr>
<tr>
<td>Banana</td>
<td>20</td>
</tr>
</tbody>
</table>
Need to be Careful…

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

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

```
SELECT quantity
FROM Purchase
```

```
SELECT product
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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>3.00</td>
<td>20</td>
</tr>
<tr>
<td>Bagel</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>0.50</td>
<td>50</td>
</tr>
<tr>
<td>Banana</td>
<td>2.00</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>4.00</td>
<td>10</td>
</tr>
</tbody>
</table>

SQLite is WRONG on the second query. Better DBMS (e.g. SQL Server) gives an error.
SQL Aside: 3-Valued Logic

```
SELECT count(*) FROM Purchase
WHERE price = 5.00 AND
price < 5.00 AND price > 5.00
```

```
SELECT count(*) FROM Purchase
WHERE price = 5.00 OR
price < 5.00 OR price > 5.00
```

Is the result different between these? If so by how much?

```
SELECT count(*) FROM Purchase
WHERE price is NULL
```
SQL Aside: 3-Valued Logic

- SQL has 3-valued logic
  - FALSE = 0 (ex. price<25 is FALSE when price = 99)
  - UNKNOWN = 0.5 (ex. price <25 is UNKNOWN when price=NULL)
  - TRUE = 1 (ex. price<25 is TRUE when price = 19)

<table>
<thead>
<tr>
<th>Logical Operation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1 AND c2</td>
<td>means min(c1,c2)</td>
</tr>
<tr>
<td>c1 OR c2</td>
<td>means max(c1,c2)</td>
</tr>
<tr>
<td>not c</td>
<td>means 1 - c</td>
</tr>
</tbody>
</table>

For SELECT ... FROM ... WHERE C do the following:
  - if C = TRUE then include the row in the output
  - if C = FALSE or C = unknown then do not include it
SQL Aside: 3-Valued Logic

Find employees who made more than 1200

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denise</td>
<td>1000</td>
<td>100</td>
</tr>
<tr>
<td>Larry</td>
<td>1000</td>
<td>500</td>
</tr>
<tr>
<td>Sharon</td>
<td>1800</td>
<td>NULL</td>
</tr>
<tr>
<td>Lawrence</td>
<td>1000</td>
<td>NULL</td>
</tr>
</tbody>
</table>

SELECT name, salary + bonus as take_home
FROM Employ
WHERE take_home > 1200

Only return Larry (1800 + NULL > 1200 is UNKNOWN)

SELECT name, salary + coalesce(bonus, 0) as take_home
FROM Employ
WHERE take_home > 1200
SQL Aside: 3-Valued Logic

Employ

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
<th>Bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denise</td>
<td>1000</td>
<td>100</td>
</tr>
<tr>
<td>Larry</td>
<td>1000</td>
<td>500</td>
</tr>
<tr>
<td>Sharon</td>
<td>1800</td>
<td>NULL</td>
</tr>
<tr>
<td>Lawrence</td>
<td>1000</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Because SQL ̲$(\ツ)_/\̲

The rule for ORDER By and GROUP BY is:
NULL values are the same (grouped together)
NULL values are less than anything else
Ordering Results

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

Note: some SQL engines want you to say ORDER BY sum(price*quantity)
Same query as earlier: (Products costing more than $1). But now, only want ones with at least 30 sales.

<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>
HAVING Clause

Same query as earlier, except that we consider only products that had at least 30 sales.

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

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 displayed as “Rev”
Show only months with less than 10 items sold
Order by quantity sold and display as “Sold”

```
SELECT month, sum(price*quantity) Rev,
       sum(quantity) as Sold
FROM  Purchase
GROUP BY month
HAVING Sold < 10
ORDER BY sum(quantity)
```
Group By and Projection

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

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

```
SELECT DISTINCT month FROM Purchase
```

Lesson: DISTINCT is a special case of GROUP BY
Let's figure out what these mean…

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

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

```
SELECT S
FROM R₁, ..., Rₙ
WHERE C₁
GROUP BY g₁, ..., gₖ
HAVING C₂
ORDER BY a₁, ..., aₙ
```

S = may contain attributes g₁, ..., gₖ and/or any aggregates but NO OTHER ATTRIBUTES
C₁ = is any condition on the attributes in R₁, ..., Rₙ
C₂ = is any condition on aggregate expressions and on attributes g₁, ..., gₖ
Semantics for SFWGHOS

```
SELECT S
FROM R_1, ..., R_n
WHERE C1
GROUP BY g_1, ..., g_k
HAVING C2
ORDER BY a_1, ..., a_n
```

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

Execution order: FWGHOS
Aggregate + Join Example

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

What do these queries mean?

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

Product(pid,pname,manufacturer)
Purchase(pid,product,price,quantity,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, count(*) 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 count(quantity)? Why not count(*)?

```
COUNT(*) includes nulls
```
Purchase(pid, product, price, quantity, month)
Product(pid, pname, manufacturer)

Exercise 1

Find all manufacturers who have sold more than 10 items (of any product). Return manufacturer name and number of items sold (as sales).

```
Select manufacture, sum(quantity) as sales
From Purchase, Product
Where product = pname
Group By manufactures
Having sales > 10
```
Exercise 1

Find all manufacturers who have sold more than 10 items (of any product). Return manufacturer name and number of items sold (as sales).

```
SELECT manufacturer, sum(quantity) as sales
FROM Product, Purchase
WHERE pname = product
GROUP BY manufacturer
HAVING sum(quantity) > 10
```
Exercise 2

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) as products
```
Exercise 2

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 3

Find all products with more than 2 purchases. Return the name of the product and max price it was sold.
Exercise 3

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

```sql
SELECT product, max(price)
FROM Purchase
JOIN Product
WHERE Purch.productid = Prod.productid
GROUP BY product
HAVING COUNT(*) > 2
```
Exercise 4

Find all manufacturers with at least 5 purchases in one month
Return manufacturer name, month, and number of items sold
Exercise 4

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) as sold
FROM Product, Purchase
GROUP BY manufacturer, month
HAVING count(*) > 5
```