COMP 430
Intro. to Database Systems
Grouping & Aggregation

Get clickers today!

Slides use ideas from Chris Ré and Chris Jermaine.
One form of aggregation – Column totals

```
SELECT Sum(price) 
FROM   Candy;
```

<table>
<thead>
<tr>
<th>candy_name</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reese’s Cup</td>
<td>0.50</td>
</tr>
<tr>
<td>5th Avenue</td>
<td>1.20</td>
</tr>
<tr>
<td>Almond Joy</td>
<td>0.75</td>
</tr>
<tr>
<td>Jelly Babies</td>
<td>2.39</td>
</tr>
</tbody>
</table>

4.84
Other aggregations

- Count
- Avg
- Max
- Min
- ...

What else depends on SQL version.
Aggregation + DISTINCT

```
SELECT Count(price), Count (DISTINCT price)
FROM   Candy;
```

<table>
<thead>
<tr>
<th>candy_name</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reese’s Cup</td>
<td>0.50</td>
</tr>
<tr>
<td>5th Avenue</td>
<td>1.20</td>
</tr>
<tr>
<td>Almond Joy</td>
<td>0.75</td>
</tr>
<tr>
<td>Jelly Babies</td>
<td>2.39</td>
</tr>
<tr>
<td>Chocolate Orange</td>
<td>2.39</td>
</tr>
<tr>
<td>Jelly Nougats</td>
<td>0.50</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>
### Aggregation & NULL

#### SELECT Count(price) FROM Candy;

<table>
<thead>
<tr>
<th>candy_name</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reese's Cup</td>
<td>0.50</td>
</tr>
<tr>
<td>5th Avenue</td>
<td>NULL</td>
</tr>
<tr>
<td>Almond Joy</td>
<td>0.75</td>
</tr>
<tr>
<td>Jelly Babies</td>
<td>2.39</td>
</tr>
<tr>
<td>Chocolate Orange</td>
<td>2.39</td>
</tr>
<tr>
<td>Jelly Nougats</td>
<td>NULL</td>
</tr>
</tbody>
</table>

No price since no longer made in original form.
Counting rows

```sql
SELECT Count(*)
FROM   Candy;
```

<table>
<thead>
<tr>
<th>candy_name</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reese’s Cup</td>
<td>0.50</td>
</tr>
<tr>
<td>5th Avenue</td>
<td>NULL</td>
</tr>
<tr>
<td>Almond Joy</td>
<td>0.75</td>
</tr>
<tr>
<td>Jelly Babies</td>
<td>2.39</td>
</tr>
<tr>
<td>Chocolate Orange</td>
<td>2.39</td>
</tr>
<tr>
<td>Jelly Nougats</td>
<td>NULL</td>
</tr>
</tbody>
</table>

6
Aggregation + other previous features

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

<table>
<thead>
<tr>
<th>product</th>
<th>date</th>
<th>price</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>bagel</td>
<td>10/21</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>banana</td>
<td>10/03</td>
<td>0.50</td>
<td>10</td>
</tr>
<tr>
<td>banana</td>
<td>10/10</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>bagel</td>
<td>10/25</td>
<td>1.50</td>
<td>20</td>
</tr>
</tbody>
</table>

50
Grouping + Aggregation – Column subtotals

```
SELECT product, Sum(price * quantity) AS subtotal
FROM   Purchase
GROUP BY product;
```
Grouping + Aggregation

```
SELECT product,  
  Count(*) AS purchases,  
  Sum(quantity) AS items,  
  Sum(price * quantity) AS subtotal  
FROM   Purchase  
GROUP BY product;
```

Purchase

<table>
<thead>
<tr>
<th>product</th>
<th>date</th>
<th>price</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>bagel</td>
<td>10/21</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>banana</td>
<td>10/03</td>
<td>0.50</td>
<td>10</td>
</tr>
<tr>
<td>banana</td>
<td>10/10</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>bagel</td>
<td>10/25</td>
<td>1.50</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>product</th>
<th>purchases</th>
<th>items</th>
<th>subtotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>bagel</td>
<td>2</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>banana</td>
<td>2</td>
<td>20</td>
<td>15</td>
</tr>
</tbody>
</table>
Conditions on aggregates

```sql
SELECT product, SUM(price * quantity) AS subtotal
FROM Purchase
WHERE date > '10/05'
GROUP BY product
HAVING SUM(quantity) >= 10;
```

<table>
<thead>
<tr>
<th>product</th>
<th>date</th>
<th>price</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>bagel</td>
<td>10/21</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>banana</td>
<td>10/03</td>
<td>0.50</td>
<td>10</td>
</tr>
<tr>
<td>banana</td>
<td>10/10</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>bagel</td>
<td>10/25</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>apple</td>
<td>10/10</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

Condition on individual rows.
Condition on aggregates.

<table>
<thead>
<tr>
<th>product</th>
<th>subtotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>bagel</td>
<td>50</td>
</tr>
<tr>
<td>banana</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>product</th>
<th>date</th>
<th>price</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>bagel</td>
<td>10/21</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>bagel</td>
<td>10/25</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>banana</td>
<td>10/10</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>apple</td>
<td>10/10</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>product</th>
<th>subtotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>bagel</td>
<td>50</td>
</tr>
<tr>
<td>banana</td>
<td>10</td>
</tr>
<tr>
<td>apple</td>
<td>5</td>
</tr>
</tbody>
</table>
Query: Products with at least two purchases.

A. Count(product)
B. Count(*)
C. Sum(product)
D. product

SELECT product
FROM Purchase
GROUP BY product
HAVING ??? >= 2 ;
SELECT product, Sum(price * quantity) 
FROM   Purchase 
GROUP BY product 
ORDER BY Sum(price * quantity);
Sorting on aggregates

```sql
SELECT product, Sum(price * quantity) AS subtotal
FROM Purchase
GROUP BY product
ORDER BY subtotal;
```

<table>
<thead>
<tr>
<th>product</th>
<th>date</th>
<th>price</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>bagel</td>
<td>10/21</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>banana</td>
<td>10/03</td>
<td>0.50</td>
<td>10</td>
</tr>
<tr>
<td>banana</td>
<td>10/10</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>bagel</td>
<td>10/25</td>
<td>1.50</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>product</th>
<th>date</th>
<th>price</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>banana</td>
<td>10/03</td>
<td>0.50</td>
<td>10</td>
</tr>
<tr>
<td>bagel</td>
<td>10/10</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>banana</td>
<td>10/25</td>
<td>1.50</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>product</th>
<th>subtotal</th>
</tr>
</thead>
<tbody>
<tr>
<td>banana</td>
<td>15</td>
</tr>
<tr>
<td>bagel</td>
<td>50</td>
</tr>
</tbody>
</table>
Syntax summary

```
SELECT [DISTINCT] Computations
FROM Tables
WHERE Condition
GROUP BY a, ..., a_k
HAVING Condition
ORDER BY attributes
LIMIT n;
```

- Uses attributes $a_1, ..., a_k$ or aggregates.
- Condition on attributes.
- Condition on aggregates.
- Can also use computation results.

Some SQLs: SELECT TOP(n) Computations
Semantics summary

SELECT [DISTINCT] Computations
FROM Tables
WHERE Condition$_1$
GROUP BY a$_1$, ..., a$_k$
HAVING Condition$_2$
ORDER BY attributes
LIMIT n;

5. Computations, [eliminate duplicates]
8. Projections
1. Cross-product
2. Apply Condition$_1$
3. Group by attributes
4. Apply Condition$_2$ to each group
6. Sort
7. Use first n rows

What we’re they thinking?!?
(Potentially confusing) details
DISTINCT Sum() vs. Sum(DISTINCT)

```
SELECT DISTINCT  Sum(DISTINCT value)
FROM         Data
GROUP BY     tag;
```

<table>
<thead>
<tr>
<th>item</th>
<th>tag</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>c</td>
<td>4</td>
</tr>
</tbody>
</table>
**HAVING without grouping**

Without GROUP BY, the entire results form one group.

```sql
SELECT value, COUNT(*) AS count FROM Data HAVING count > 1;
```

<table>
<thead>
<tr>
<th>item</th>
<th>tag</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>c</td>
<td>4</td>
</tr>
</tbody>
</table>

Fails in SQLite.
Grouping without aggregation

```
SELECT product
FROM   Purchase
GROUP BY product;
```

<table>
<thead>
<tr>
<th>product</th>
<th>date</th>
<th>price</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>bagel</td>
<td>10/21</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>banana</td>
<td>10/03</td>
<td>0.5</td>
<td>10</td>
</tr>
<tr>
<td>banana</td>
<td>10/10</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>bagel</td>
<td>10/25</td>
<td>1.50</td>
<td>20</td>
</tr>
</tbody>
</table>
GROUP BY vs. DISTINCT

SELECT product
FROM   Purchase
GROUP BY  product;

SELECT DISTINCT  product
FROM   Purchase;

But only GROUP BY works well with aggregation.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>product</td>
<td>date</td>
<td>price</td>
<td>quantity</td>
</tr>
<tr>
<td>bagel</td>
<td>10/21</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>banana</td>
<td>10/03</td>
<td>0.5</td>
<td>10</td>
</tr>
<tr>
<td>banana</td>
<td>10/10</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>bagel</td>
<td>10/25</td>
<td>1.50</td>
<td>20</td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>product</td>
</tr>
<tr>
<td>bagel</td>
</tr>
<tr>
<td>banana</td>
</tr>
</tbody>
</table>
More complicated examples
Activity – will break activity into segments

04a-aggregation.ipynb

- Students (id and name) taking more than 5 courses
- Courses and their average ratings
- Highest average rating of a course
- Course with highest average rating
Students with >5 courses – Two solutions

```
SELECT s_id, first_name, last_name
FROM   Student
WHERE (SELECT Count(*)
       FROM  Enrollment
       WHERE Student.s_id = Enrollment.s_id
    ) > 5;
```

```
SELECT  s.s_id, s.first_name, s.last_name
FROM   Student s, Enrollment e
WHERE s.s_id = e.s_id
GROUP BY Student.s_id
HAVING COUNT(crn) > 5;
```
Efficiency comparison

Assuming no major optimizations:

1. How many times do we SELECT ... FROM Enrollment?
2. How many joins?

```sql
SELECT s_id, first_name, last_name
FROM   Student
WHERE (SELECT Count(*)
       FROM  Enrollment
       WHERE Student.s_id = Enrollment.s_id
       ) > 5;

SELECT s.s_id, s.first_name, s.last_name
FROM   Student s, Enrollment e
WHERE s.s_id = e.s_id
GROUP BY Student.s_id
HAVING  COUNT(crn) > 5;
```
Courses and their average ratings

```
SELECT crn, Avg(rating)
FROM   Enrollment
GROUP BY crn;
```
Highest average rating of a course

\[
\begin{align*}
\text{SELECT} & \hspace{1em} \text{crn, Max(Avg(rating))} \\
\text{FROM} & \hspace{1em} \text{Enrollment} \\
\text{GROUP BY} & \hspace{1em} \text{crn}; \\
\end{align*}
\]

Nesting aggregate functions is not syntactically allowed.

\[
\begin{align*}
\text{SELECT} & \hspace{1em} \text{Max(avg\_rating)} \\
\text{FROM} & \hspace{1em} (\text{SELECT} \hspace{1em} \text{Avg(rating) as avg\_rating} \\
& \hspace{2em} \text{FROM} \hspace{1em} \text{Enrollment} \\
& \hspace{2em} \text{GROUP BY} \hspace{1em} \text{crn})
\end{align*}
\]
SELECT crn
FROM (SELECT crn, Avg(rating) AS avg_rating1
     FROM Enrollment
     GROUP BY crn
     )
WHERE avg_rating1 = (SELECT Max(avg_rating2)
                      FROM (SELECT Avg(rating) as avg_rating2
                             FROM Enrollment
                             GROUP BY crn
                             )
                      );

1. Calculate all average ratings.
2. Calculate the maximum.
3. Find which courses have this value as their average.
What about repeated subquery?

```
SELECT crn
FROM   (SELECT crn, Avg(rating) AS avg_rating1
           FROM   Enrollment
           GROUP BY crn
      )
WHERE  avg_rating1 = (SELECT  Max(avg_rating2)
                        FROM    (SELECT Avg(rating) AS avg_rating2
                                   FROM   Enrollment
                                   GROUP BY crn
                              )
                      );
```

```
CREATE VIEW Average AS
SELECT crn, Avg(rating) AS avg_rating
FROM   Enrollment
GROUP BY crn;

SELECT crn
FROM   Average
WHERE  avg_rating = (SELECT  Max(avg_rating)
                      FROM   Average
                      );
```
Highest avg rating – Solution 2

CREATE VIEW Average AS
SELECT crn, Avg(rating) AS avg_rating
FROM Enrollment
GROUP BY crn;

SELECT crn
FROM Average
ORDER BY avg_rating DESC
LIMIT 1;

 Doesn’t work for ties.
Efficiency comparison

CREATE VIEW Averages AS
SELECT crn, Avg(rating) AS avg_rating
FROM Enrollment
GROUP BY crn;

SELECT crn
FROM Averages
WHERE avg_rating = (SELECT Max(avg_rating)
FROM Averages);

SELECT crn
FROM Averages
ORDER BY avg_rating
LIMIT 1;

Assuming no major optimizations:
Cost estimate of each?
Aside: Code Style & Indentation

Review previous examples.
Where we’re headed

• So far:
  • Have seen a powerful subset of SQL queries.
  • Raised questions about how example schemas are structured.

• Next:
  • Switch to focusing on database design.

• Later:
  • Return to more SQL.