Introduction to Database Systems
CSE 444

Lecture 3: SQL (part 2)
Outline

- Aggregations (6.4.3 – 6.4.6)
- Examples, examples, examples...
- Nulls (6.1.6 - 6.1.7) [Old edition: 6.1.5-6.1.6]
- Outer joins (6.3.8)
Aggregation

SELECT avg(price)  
FROM Product  
WHERE maker='Toyota'

SELECT count(*)  
FROM Product  
WHERE year > 1995

SQL supports several aggregation operations:

sum, count, min, max, avg

Except count, all aggregations apply to a single attribute
Aggregation: Count Distinct

COUNT applies to duplicates, unless otherwise stated:

```
SELECT count (category)  # same as Count(*)
FROM Product
WHERE year > 1995
```

We probably want:

```
SELECT count (DISTINCT category)  
FROM Product
WHERE year > 1995
```
Simple Aggregation 1/2

Purchase (product, price, quantity)

SELECT sum (price * quantity) FROM Purchase

SELECT sum (price * quantity) FROM Purchase WHERE product = 'Bagel'

What do these queries mean?
### Simple Aggregation 2/2

#### Purchase

<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>2</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>1</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**  
`sum (price * quantity)`  
**FROM**  
Purchase  
**WHERE**  
`product = 'Bagel'`

3 * 20 = 60  
2 * 20 = 40  

**sum**: 100

SQL creates attribute name

http://www.cs.washington.edu/education/courses/cse444/11wi/
Find total quantities for all sales over $1, by product.
From → Where → Group By → 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>2</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>1</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 contains
- grouped attributes
- and aggregates

```
SELECT product, sum(quantity) as TotalSales
FROM Purchase
WHERE price > 1
GROUP BY product
```
Group By v.s. Nested Queries

\[
\begin{align*}
\text{SELECT} & \quad \text{DISTINCT} \quad x.\text{product}, \\
( & \quad \text{SELECT} \quad \text{sum}(y.\text{quantity}) \\
\text{FROM} & \quad \text{Purchase} \quad y \\
\text{WHERE} & \quad x.\text{product} = y.\text{product} \\
\text{and} & \quad \text{price} > 1) \quad \text{as} \quad \text{TotalSales} \\
\text{FROM} & \quad \text{Purchase} \quad x \\
\text{WHERE} & \quad \text{price} > 1
\end{align*}
\]

Why twice?

\[
\begin{align*}
\text{SELECT} & \quad \text{product}, \quad \text{sum}(\text{quantity}) \quad \text{as} \quad \text{TotalSales} \\
\text{FROM} & \quad \text{Purchase} \\
\text{WHERE} & \quad \text{price} > 1 \\
\text{GROUP BY} & \quad \text{product}
\end{align*}
\]

http://www.cs.washington.edu/education/courses/cse444/11wi/
Another Example

```sql
SELECT product,
      sum(quantity) as SumQuantity,
      max(price) as MaxPrice
FROM Purchase
GROUP BY product
```

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
<th>MaxPrice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>40</td>
<td>3</td>
</tr>
<tr>
<td>Banana</td>
<td>70</td>
<td>4</td>
</tr>
</tbody>
</table>

Next, focus only on products with at least 50 sales
HAVING Clause

Q: Similar to before, but only products with at least 30 sales.

```sql
SELECT product, 
    sum(quantity) as SumQuantity, 
    max(price) as MaxPrice 
FROM Purchase 
GROUP BY product 
HAVING sum(quantity) > 50
```

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
<th>MaxPrice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banana</td>
<td>70</td>
<td>4</td>
</tr>
</tbody>
</table>
General form of Grouping and Aggregation

1. Evaluate From → Where, apply condition C1
2. Group by the attributes a₁,...,aₖ
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result
Advanced SQLizing

1. Getting around INTERSECT and EXCEPT
2. Unnesting Aggregates
3. Finding witnesses
INTERSECT and EXCEPT*

\[
\text{INTERSECT} \quad (\text{SELECT R.A, R.B FROM R}) \quad \text{INTERSECT} \quad (\text{SELECT S.A, S.B FROM S})
\]

\[
\text{SELECT R.A, R.B FROM R WHERE EXISTS (SELECT * FROM S WHERE R.A=S.A and R.B=S.B)}
\]

Can be unnested. How?

\[
\text{EXCEPT} \quad (\text{SELECT R.A, R.B FROM R}) \quad \text{EXCEPT} \quad (\text{SELECT S.A, S.B FROM S})
\]

\[
\text{SELECT R.A, R.B FROM R WHERE NOT EXISTS (SELECT * FROM S WHERE R.A=S.A and R.B=S.B)}
\]

*Not in all DBMSs
Unnesting Aggregates

Product (pname, price, company)
Company (cname, city)

Find the number of companies in each city

\[
\begin{align*}
\text{SELECT DISTINCT} & \quad \text{city, } (\text{SELECT count(*)} \\
& \quad \text{FROM Company Y} \\
& \quad \text{WHERE Y.city = X.city}) \\
\text{FROM} & \quad \text{Company X}
\end{align*}
\]

Equivalent queries (as long as no NULLs)

\[
\begin{align*}
\text{SELECT} & \quad \text{city, count(*)} \\
\text{FROM} & \quad \text{Company} \\
\text{GROUP BY} & \quad \text{city}
\end{align*}
\]

Note: no need for DISTINCT (DISTINCT \textit{is the same} as GROUP BY)
Unnesting Aggregates

Product ( pname, price, company)  
Company(cname, city)

Find the number of products made in each city

\[
\begin{align*}
\text{SELECT DISTINCT } & \text{X.city, (SELECT count(*)} \\
& \text{FROM Product Y, Company Z} \\
& \text{WHERE Z.cname = Y.company} \\
& \text{and Z.city = X.city}) \\
\text{FROM Company X}
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \text{ X.city, count(*)} \\
\text{FROM} & \text{ Company X, Product Y} \\
\text{WHERE} & \text{ X.cname = Y.company} \\
\text{GROUP BY} & \text{ X.city}
\end{align*}
\]

What if there are no products for a city? 😊

They are not equivalent!! Why??

http://www.cs.washington.edu/education/courses/cse444/11wi/
More on Unnesting

- Find all authors who wrote at least 10 documents:

  ```sql
  SELECT DISTINCT Author.name
  FROM Author
  WHERE (SELECT count(Wrote.url)
         FROM Wrote
         WHERE Author.login=Wrote.login)
     > 10
  ```

  Works, but this is bad style

- Second attempt (no nesting):

  ```sql
  SELECT Author.name
  FROM Author, Wrote
  WHERE Author.login=Wrote.login
  GROUP BY Author.name
  HAVING count(Wrote.url) > 10
  ```

  Much better!
Finding Witnesses

Store(sid, sname)
Product(pid, pname, price, sid)

Q: For each store, find its most expensive products

Finding the maximum price is easy…

```
SELECT Store.sid, max(Product.price)
FROM Store, Product
WHERE Store.sid = Product.sid
GROUP BY Store.sid
```

But we want the “witnesses”, i.e. the products with max price
Finding Witnesses

Plan:

- Compute max price in a subquery
- Compare it with each product price

```sql
SELECT Store.sname, Product.pname
FROM Store, Product,
     (SELECT Store.sid as sid, max(Product.price) as p
      FROM Store, Product
      WHERE Store.sid = Product.sid
      GROUP BY Store.sid) X
WHERE Store.sid = Product.sid
    AND Store.sid = X.sid
    AND Product.price = X.p
```
Finding Witnesses

There is a more concise solution here:

```
SELECT Store.sname, x.pname
FROM Store, Product x
WHERE Store.sid = x.sid
  and x.price >=
    ALL (SELECT y.price
         FROM Product y
         WHERE Store.sid = y.sid)
```
NULLS in SQL

- Whenever we don’t have a value, we can put a NULL

- Can mean many things:
  - Value does not exist
  - Value exists but is unknown
  - Value not applicable
  - Etc.

- The schema specifies for each attribute if it can be NULL (*nullable* attribute) or not

- How does SQL cope with tables that have NULLs?

http://www.cs.washington.edu/education/courses/cse444/11wi/
Null Values

- If \( x = \text{NULL} \) then
  - Arithmetic operations produce NULL. E.g: \( 4*(3-x)/7 \)
  - Boolean conditions are also NULL. E.g: \( x = 'Joe' \)

- In SQL there are three boolean values:
  - FALSE, TRUE, UNKNOWN

  - Reasoning:
    - FALSE = 0
    - TRUE = 1
    - UNKNOWN = 0.5
    - \( x \text{ AND } y = \min(x,y) \)
    - \( x \text{ OR } y = \max(x,y) \)
    - \( \text{NOT } x = (1 - x) \)
Null Values: example

**Rule in SQL:** include only tuples that yield TRUE

<table>
<thead>
<tr>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>NULL</td>
<td>200</td>
</tr>
<tr>
<td>NULL</td>
<td>6.5</td>
<td>170</td>
</tr>
</tbody>
</table>

**Unexpected behavior**

**Test NULL explicitly**
Outerjoins

Product(name, category)
Purchase(prodName, store)

An “inner join”:

```
SELECT Product.name, Purchase.store
FROM   Product, Purchase
WHERE  Product.name = Purchase.prodName
```

Same as:

```
SELECT Product.name, Purchase.store
FROM   Product JOIN Purchase ON
        Product.name = Purchase.prodName
```

But Products that never sold will be lost!
Outerjoins

If we want the never-sold products, we need an “outerjoin”:

```sql
SELECT Product.name, Purchase.store
FROM Product LEFT OUTER JOIN Purchase ON Product.name = Purchase.prodName
```

<table>
<thead>
<tr>
<th>Product</th>
<th>Purchase</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>ProdName</strong></td>
<td><strong>Store</strong></td>
</tr>
<tr>
<td>Gizmo</td>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>

Inner join does not produce this tuple
Example

Compute, for each product, the total number of sales in ‘September’

```
SELECT Product.name, count(*)
FROM Product, Purchase
WHERE Product.name = Purchase.prodName
    and Purchase.month = 'September'
GROUP BY Product.name
```

What’s wrong?
Example

Compute, for each product, the total number of sales in ‘September’

```
SELECT Product.name, count(store)
FROM Product
LEFT OUTER JOIN Purchase
ON Product.name = Purchase.prodName
and Purchase.month = 'September'
GROUP BY Product.name
```

We need to use the attribute to get the correct 0 count. (§6.4.6)

Now we also get the products with 0 sales
Outer Joins: summary

- **Left outer join:**
  - Include the left tuple even if there’s no match

- **Right outer join:**
  - Include the right tuple even if there’s no match

- **Full outer join:**
  - Include both left and right tuples even if there’s no match