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

<table>
<thead>
<tr>
<th>SELECT avg(price) FROM Product WHERE maker='Toyota'</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT count(*) FROM Product WHERE year &gt; 1995</td>
</tr>
</tbody>
</table>

SQL supports several aggregation operations:

sum, count, min, max, avg

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

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
```
More Examples

Purchase\( (\text{product, date, price, quantity}) \)

\[
\begin{align*}
\text{SELECT} & \quad \text{Sum(price} \times \text{quantity)} \\
\text{FROM} & \quad \text{Purchase}
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad \text{Sum(price} \times \text{quantity)} \\
\text{FROM} & \quad \text{Purchase} \\
\text{WHERE} & \quad \text{product} = \text{‘bagel’}
\end{align*}
\]
## Simple Aggregations

**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>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>

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

90 (= 60+30)
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
```

Let’s see what this means…
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-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>
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 TotalSales
FROM Purchase
WHERE price > 1
GROUP BY product
```
GROUP BY v.s. Nested Queries

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

```
SELECT DISTINCT x.product, (SELECT Sum(y.quantity)
FROM Purchase y
WHERE x.product = y.product
AND price > 1)
AS TotalSales
FROM Purchase x
WHERE price > 1
```

Why twice?
Another Example

What does it mean?

```
SELECT product, 
    sum(quantity) AS SumQuantity, 
    max(price) AS MaxPrice 
FROM Purchase 
GROUP BY product
```
HAVING Clause

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

```sql
SELECT product, Sum(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

```
SELECT  S  
FROM    R₁, ..., Rₙ  
WHERE   C₁  
GROUP BY a₁, ..., aₖ  
HAVING  C₂
```

S = may contain attributes a₁, ..., aₖ 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 a₁, ..., aₖ

Why?
General form of Grouping and Aggregation

SELECT S
FROM R₁,…,Rₙ
WHERE C₁
GROUP BY a₁,…,aₖ
HAVING C₂

Evaluation steps:
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: not in some DBMSs

**INTERSECT and EXCEPT:**

```
(SELECT R.A, R.B FROM R)
INTERSECT
(SELECT S.A, S.B FROM S)
```

**Can unnest.**

**How?**

```
SELECT R.A, R.B FROM R
WHERE
EXISTS(SELECT *
FROM S
WHERE R.A=S.A and R.B=S.B)
```

```
(SELECT R.A, R.B FROM R)
EXCEPT
(SELECT S.A, S.B FROM S)
```

```
SELECT R.A, R.B FROM R
WHERE
NOT EXISTS(SELECT *
FROM S
WHERE R.A=S.A and R.B=S.B)
```
Unnesting Aggregates

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

Find the number of companies in each city

```
SELECT DISTINCT city, (SELECT count(*)
    FROM Company Y
    WHERE X.city = Y.city)
FROM Company X
```

```
SELECT city, count(*)
FROM Company
GROUP BY city
```

Equivalent queries

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

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

Find the number of products made in each city

SELECT DISTINCT X.city, (SELECT count(*)
FROM Product Y, Company Z
WHERE Z.cname=Y.company
AND Z.city = X.city)
FROM Company X

SELECT X.city, count(*)
FROM Company X, Product Y
WHERE X.cname=Y.company
GROUP BY X.city

What if there are no products for a city?

They are NOT equivalent! (WHY?)
More Unnesting

Author(login,name)
Wrote(login,url)

- Find authors who wrote ≥ 10 documents:
- Attempt 1: with nested queries

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

This is SQL by a novice
More Unnesting

• Find all authors who wrote at least 10 documents:
• Attempt 2: SQL style (with GROUP BY)

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

This is SQL by an expert
Finding Witnesses

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

For each store,
find its most expensive products
Finding Witnesses

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 need the *witnesses*, i.e. the products with max price
Finding Witnesses

To find the witnesses, compute the maximum price in a subquery

```
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 exists
  – Value exists but is unknown
  – Value not applicable
  – Etc.
• The schema specifies for each attribute if can be null (nullable attribute) or not
• How does SQL cope with tables that have NULLs?
Null Values

• If x= NULL then 4*(3-x)/7 is still NULL

• If x= NULL then x=‘Joe’ is UNKNOWN

• In SQL there are three boolean values:

  FALSE     = 0
  UNKNOWN   = 0.5
  TRUE      = 1
Null Values

- \( C_1 \text{ AND } C_2 = \min(C_1, C_2) \)
- \( C_1 \text{ OR } C_2 = \max(C_1, C_2) \)
- \( \text{NOT } C_1 = 1 - C_1 \)

```sql
SELECT *
FROM Person
WHERE (age < 25) AND (height > 6 OR weight > 190)
```

Rule in SQL: include only tuples that yield TRUE

E.g.
- age=20
- height=NULL
- weight=200
Null Values

Unexpected behavior:

```
SELECT *
FROM Person
WHERE age < 25 OR age >= 25
```

Some Person tuples are not included!
Null Values

Can test for NULL explicitly:

- x IS NULL
- x IS NOT NULL

```
SELECT *
FROM Person
WHERE age < 25  OR  age >= 25 OR age IS NULL
```

Now it includes all Person tuples
**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

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

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
</tbody>
</table>

### Purchase

<table>
<thead>
<tr>
<th>ProdName</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
<tr>
<td>OneClick</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Application

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

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

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

What’s wrong?
Application

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

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

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

Now we also get the products who sold in 0 quantity
Outer Joins

• 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