Lecture 03: SQL

Friday, March 30, 2007
Outline

• Subqueries (6.3)
• Aggregations (6.4.3 – 6.4.6)
• Examples, examples, examples…

Read the entire chapter 6!
Aggregation

\[
\text{SELECT } \text{avg} \text{(price)} \text{ FROM Product WHERE maker="Toyota"}
\]
\[
\text{SELECT } \text{count}(\ast) \text{ FROM Product WHERE year > 1995}
\]

SQL supports several aggregation operations:

sum, count, min, max, avg

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

COUNT applies to duplicates, unless otherwise stated:

```
SELECT Count(category) FROM Product WHERE year > 1995
```

same as Count(*)

We probably want:

```
SELECT Count(DISTINCT category) FROM Product WHERE year > 1995
```
More Examples

Purchase(product, date, price, quantity)

```
SELECT Sum(price * quantity) 
FROM Purchase
```

```
SELECT Sum(price * quantity) 
FROM Purchase 
WHERE product = 'bagel'
```
Simple Aggregations

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>10/3</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>

\[
\text{SELECT } \text{Sum(price * quantity)} \\
\text{FROM } \text{Purchase} \\
\text{WHERE product = 'bagel'}
\]

50 \ (= 20+30)
Grouping and Aggregation

Purchase(product, date, price, quantity)

Find total sales after 10/1/2005 per product.

```
SELECT product, Sum(price*quantity) AS TotalSales
FROM Purchase
WHERE date > '10/1/2005'
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>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>10/3</td>
<td>0.5</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>10/10</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>
### 3. SELECT

```
SELECT product, Sum(price*quantity) AS TotalSales
FROM Purchase
WHERE date > '10/1/2005'
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>Bagel</td>
<td>10/25</td>
<td>1.50</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>10/3</td>
<td>0.5</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>10/10</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>50</td>
</tr>
<tr>
<td>Banana</td>
<td>15</td>
</tr>
</tbody>
</table>
GROUP BY v.s. Nested Quereis

```sql
SELECT product, Sum(price*quantity) AS TotalSales
FROM Purchase
WHERE date > '10/1/2005'
GROUP BY product
```

```sql
SELECT DISTINCT x.product, (SELECT Sum(y.price*y.quantity)
FROM Purchase y
WHERE x.product = y.product
AND y.date > '10/1/2005')
AS TotalSales
FROM Purchase x
WHERE x.date > '10/1/2005'
```
Another Example

```
SELECT product, 
    sum(price * quantity) AS SumSales 
    max(quantity) AS MaxQuantity
FROM Purchase 
GROUP BY product
```

What does it mean?
HAVING Clause

Same query, except that we consider only products that had at least 100 buyers.

```
SELECT product, Sum(price * quantity) 
FROM Purchase 
WHERE date > '10/1/2005' 
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
General form of Grouping and Aggregation

\[
\text{SELECT } S \\
\text{FROM } R_1, \ldots, R_n \\
\text{WHERE } C_1 \\
\text{GROUP BY } a_1, \ldots, a_k \\
\text{HAVING } C_2
\]

Evaluation steps:
1. Evaluate FROM-WHERE, apply condition C1
2. Group by the attributes \( a_1, \ldots, a_k \)
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. Quantifiers
3. Aggregation v.s. subqueries
4. Two examples (study at home)
1. INTERSECT and EXCEPT:

\[
\begin{align*}
\text{(SELECT } R.A, R.B \text{ FROM } R) \\
\text{ INTERSECT } \\
\text{(SELECT } S.A, S.B \text{ FROM } S) \\
\end{align*}
\]

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

\[
\begin{align*}
\text{(SELECT } R.A, R.B \text{ FROM } R) \\
\text{ EXCEPT } \\
\text{(SELECT } S.A, S.B \text{ FROM } S) \\
\end{align*}
\]

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

If R, S have no duplicates, then can write without subqueries (HOW?)

INTERSECT and EXCEPT: not in SQL Server
2. Quantifiers

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

Find all companies that make some products with price < 100

```
SELECT DISTINCT Company.cname 
FROM Company, Product 
WHERE Company.cname = Product.company and Product.price < 100
```

Existential: easy 😊
2. Quantifiers

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

Find all companies that make only products with price < 100

same as:

Find all companies s.t. all of their products have price < 100

Universal: hard ! 😞
2. Quantifiers

1. Find the other companies: i.e. s.t. some product ≥ 100

```sql
SELECT DISTINCT Company.cname
FROM Company
WHERE Company.cname IN (SELECT Product.company
FROM Product
WHERE Product.price >= 100)
```

2. Find all companies s.t. all their products have price < 100

```sql
SELECT DISTINCT Company.cname
FROM Company
WHERE Company.cname NOT IN (SELECT Product.company
FROM Product
WHERE Product.price >= 100)
```
3. Group-by v.s. Nested Query

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

• Find authors who wrote $\geq 10$ documents:
• Attempt 1: with nested queries

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

This is SQL by a novice
3. Group-by v.s. Nested Query

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

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

No need for **DISTINCT**: automatically from **GROUP BY**
3. Group-by v.s. Nested Query

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

Find authors with vocabulary ≥ 10000 words:

```
SELECT Author.name
FROM Author, Wrote, Mentions
WHERE Author.login=Wrote.login AND Wrote.url=Mentions.url
GROUP BY Author.name
HAVING count(distinct Mentions.word) > 10000
```
4. Two Examples

\[
\text{Store}(\text{sid, sname}) \\
\text{Product}(\text{pid, pname, price, sid})
\]

Find all stores that sell \textit{only} products with price > 100

same as:

Find all stores s.t. all their products have price > 100)
Almost equivalent…

```
SELECT Store.name
FROM Store
WHERE 100 < ALL (SELECT Product.price
                  FROM product
                  WHERE Store.sid = Product.sid)
```

```
SELECT Store.name
FROM Store
WHERE Store.sid NOT IN (SELECT Product.sid
                         FROM Product
                         WHERE Product.price <= 100)
```

```
SELECT Store.name FROM Store
WHERE 100 < min(Product.price)
GROUP BY Store.sid, Store.name
HAVING 100 < min(Product.price)
```

Why both?
Two Examples

\[\text{Store}(\text{sid, sname})\]
\[\text{Product}(\text{pid, pname, price, sid})\]

For each store,
find its most expensive product
Two Examples

This is easy but doesn’t do what we want:

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

Better:

```sql
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)
```

But may return multiple product names per store
Two Examples

Finally, choose some pid arbitrarily, if there are many with highest price:

```sql
SELECT Store.sname, max(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)
GROUP BY Store.sname
```