Introduction to Database Systems  
CSE 444  
Lecture 03: SQL  
October(!) 1, 2007

Outline

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

Read the entire chapter 6!

Aggregation

```
SELECT avg(price) FROM Product WHERE maker="Toyota"
```

SQL supports several aggregation operations:

- sum, count, min, max, avg

Except count, all aggregations apply to a single attribute

```
SELECT Count(category) FROM Product
```

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

Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

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

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

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’
```

What do they mean?

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

```
SELECT Sum(price * quantity) FROM Purchase WHERE product = ‘bagel’
```

Simple Aggregations

```
<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>
```

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

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
```

GROUP BY v.s. Nested Queries

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

```
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
```
HAVING Clause

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

```sql
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

```sql
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

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. Quantifiers
3. Aggregation v.s. subqueries
4. Two examples (study at home)

1. INTERSECT and EXCEPT: not in SQL Server

1. INTERSECT and EXCEPT:

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

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

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

2. Quantifiers

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

Find all companies that make some products with price < 100

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

Existential: easy ! ☺
2. Quantifiers

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

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

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 ≥ 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

• 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

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

Find all stores that sell only products with price > 100
same as:
Find all stores s.t. all their products have price > 100)

This is SQL by a novice

This is SQL by an expert
Two Examples

For each store, find its most expensive product

Two Examples

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

Better:

But may return multiple product names per store

Two Examples

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