Practice and Applications of Data Management

CMPSCI 345

Lecture 05: Advanced Joins and Aggregates
Today:

- Continue on joins
  - Self-join
  - Outer-join

- Aggregates
To-dos

- Gradiance quizzes:
  - Lab Quiz is due on Monday
    - Slightly different form: you write queries that are checked against a sample dataset
  - Quiz 3 is due next Wednesday

- Homework assignment 1 is posted
  - Moodle to-dos can pace you through it

- Install Postgres
Example 6 from hand-out

- Return a list of all employee names and the names of their managers

```
SELECT e1.name, e2.name AS manager
FROM Employees as e1, Employees as e2
WHERE e1.managerID = e2.empID
```
Practice self-joins

- Return a list of all employee names who work on both the ‘Web archive’, and ‘Phone app’ projects

```
SELECT e.name
FROM Employees as e, Projects as p1,
     Projects as p2
WHERE e.empID = p1.empID
     AND e.empID = p2.empID
     AND p1.project='Web archive'
     AND p2.project='Phone app'
```
Practice self-joins

Return a list of all employee names who work on two distinct projects

```
SELECT DISTINCT e.name
FROM Employees as e, Projects as p1, Projects as p2
WHERE e.empID = p1.empID
AND e.empID = p2.empID
AND p1.project <> p2.project
```
NULLs in SQL

insert into Company values ('Apple', 'USA');
insert into Product values ('iPad 5', NULL, 'gadget', 'Apple');

```
SELECT * 
FROM   Product 
WHERE  price < 100
```

```
SELECT * 
FROM   Product 
WHERE  price >= 100
```

Where is the iPad?
NULLs in Joins

`insert into Company values('Google', 'USA');`

Another way to write a join

```sql
SELECT * 
FROM Company x, Product y 
WHERE x.cname = y.manufacturer
```

```sql
SELECT * 
FROM Company INNER JOIN Product ON 
cname = manufacturer
```
Left outer join

- How to retrieve everything:

```sql
SELECT * 
FROM Company LEFT OUTER JOIN Product ON cname = manufacturer
```
List all employee names and the projects that each is involved in. Do not omit any employees even if they don’t have a project.

```
SELECT e.name, p.project
FROM Employees as e
LEFT OUTER JOIN Projects as p
ON e.empID = p.empID
```
Practice

- Return a list of all employee names and the names of their managers. Do not omit any employees even if they have no manager.

```
SELECT e1.name, e2.name AS manager
FROM Employees as e1
    LEFT OUTER JOIN Employees as e2
    ON e1.managerID = e2.empID
```
Example: outer joins

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

```
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>Name</td>
<td>ProdName</td>
<td>Name</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Gizmo</td>
<td>Gizmo</td>
</tr>
<tr>
<td>Camera</td>
<td>Camera</td>
<td>Camera</td>
</tr>
<tr>
<td>OneClick</td>
<td>OneClick</td>
<td>OneClick</td>
</tr>
<tr>
<td>Category</td>
<td>Store</td>
<td>Store</td>
</tr>
<tr>
<td>Gadget</td>
<td>Wiz</td>
<td>Wiz</td>
</tr>
<tr>
<td>Photo</td>
<td>Ritz</td>
<td>Ritz</td>
</tr>
<tr>
<td>Photo</td>
<td>Wiz</td>
<td>Wiz</td>
</tr>
<tr>
<td>NULL</td>
<td></td>
<td>NULL</td>
</tr>
</tbody>
</table>

Inner join does not produce this tuple
Aggregation

SQL supports several aggregation operations:

- `sum`, `count`, `min`, `max`, `avg`

Except `count`, all aggregations apply to a single attribute.
COUNT applies to duplicates, unless otherwise stated:

```sql
SELECT count (category) FROM Product WHERE price > 100
```

(almost) same as `Count(*)`.

We probably want:

```sql
SELECT count (DISTINCT category) FROM Product WHERE price > 100
```
Practice count

insert into Company values('Microsoft', NULL);

- How many companies are there?
- How many countries have companies?
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>

\[
3 \times 20 = 60 \\
2 \times 20 = 40 \\
\text{sum: 100}
\]

SQL creates attribute name

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

(No column name)

100
### Grouping and Aggregation

<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>

**Find total quantities for all sales over $1, by product.**

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>40</td>
</tr>
<tr>
<td>Banana</td>
<td>20</td>
</tr>
</tbody>
</table>
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>

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>40</td>
</tr>
<tr>
<td>Banana</td>
<td>20</td>
</tr>
</tbody>
</table>

Select contains:
- grouped attributes
- and aggregates

4 SELECT product, sum(quantity) as TotalSales
1 FROM Purchase
2 WHERE price > 1
3 GROUP BY product
Example

```
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

```
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>

Q: Similar to before, but only products with at least 50 sales.
General form of Grouping and Aggregation

```
5 SELECT S
1 FROM R_1,...,R_n
2 WHERE C1
3 GROUP BY a_1,...,a_k
4 HAVING C2
```

**Evaluation**

1. Evaluate From
2. Evaluate Where: apply condition C1
3. Group by the attributes a_1,...,a_k
4. Apply condition C2 to each group (may have aggregates)
5. Compute aggregates in S and return the result
Example

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

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

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

What’s wrong?