Introduction to Database Systems
CSE 414

Lectures 4 and 5: Aggregates in SQL
Announcements

• Homework 1 is due on Wednesday
• Quiz 2 will be out today and due on Friday
Outline

• Outer joins (6.3.8)
• Aggregations (6.4.3 – 6.4.6)
• Examples, examples, examples…
Outerjoins

Product(name, category)
Purchase(prodName, store) -- prodName is foreign key

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 some Products are not listed! Why?
Outerjoins

Product(name, category)
Purchase(prodName, store) -- prodName is foreign key

If we want to include products that never sold, then we need an “outerjoin”:

```sql
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>
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
Aggregation in SQL

>sqlite3 lecture04

sqlite> create table Purchase
    (pid int primary key,
     product text,
     price float,
     quantity int,
     month varchar(15));

sqlite> -- download data.txt
sqlite> .import data.txt Purchase

Specify a filename where the database will be stored

Other DBMSs have other ways of importing data
Comment about SQLite

• One cannot load NULL values such that they are actually loaded as null values

• So we need to use two steps:
  – Load null values using some type of special value
  – Update the special values to actual null values

  ```
  update Purchase
  set price = null
  where price = ‘null’
  ```
Simple Aggregations

Five basic aggregate operations in SQL

\[
\begin{align*}
\text{select count(*) from Purchase} \\
\text{select sum(quantity) from Purchase} \\
\text{select avg(price) from Purchase} \\
\text{select max(quantity) from Purchase} \\
\text{select min(quantity) from Purchase}
\end{align*}
\]

Except count, all aggregations apply to a single attribute
Aggregates and NULL Values

Null values are not used in aggregates

```
insert into Purchase
values(12, 'gadget', NULL, NULL, 'april')
```

Let’s try the following

```
select count(*) from Purchase
select count(quantity) from Purchase

select sum(quantity) from Purchase

select sum(quantity) from Purchase
where quantity is not null;
```
COUNT applies to duplicates, unless otherwise stated:

```
SELECT Count(product)
FROM Purchase
WHERE price > 4.99
```

same as Count(*)

We probably want:

```
SELECT Count(DISTINCT product)
FROM Purchase
WHERE price > 4.99
```
More Examples

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

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

What do they mean?
### Simple Aggregations

<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>

\[
\text{SELECT } \text{Sum(price} \times \text{quantity)} \text{ FROM Purchase WHERE product = 'Bagel'} \Rightarrow 90 \ (= 60+30)
\]
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>

WHERE price > 1
### 3. SELECT

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

<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>

<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>
Other Examples

Compare these two queries:

```sql
SELECT product, count(*)
FROM Purchase
GROUP BY product
```

```sql
SELECT month, count(*)
FROM Purchase
GROUP BY month
```

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

What does it mean?
Need to be Careful…

```sql
SELECT product, max(quantity) 
FROM Purchase 
GROUP BY product
```

```sql
SELECT product, quantity 
FROM Purchase 
GROUP BY product
```

sqlite is WRONG on this query.

<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>

Advanced DBMS (e.g. SQL Server) gives an error
Ordering Results

```
SELECT product, sum(price*quantity) as rev
FROM purchase
GROUP BY product
ORDER BY rev desc
```
HAVING Clause

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

```sql
SELECT product, sum(price*quantity) 
FROM Purchase 
WHERE price > 1 
GROUP BY product 
HAVING Sum(quantity) > 30
```

HAVING clause contains conditions on aggregates.
WHERE vs HAVING

- **WHERE** condition is applied to individual rows
  - The rows may or may not contribute to the aggregate
  - No aggregates allowed here

- **HAVING** condition is applied to the entire group
  - Entire group is returned, or not at all
  - May use aggregate functions in the group
create table Product
    (pid int primary key,
     pname varchar(15),
     manufacturer varchar(15));

insert into product values(1,"bagel","Sunshine Co.");
insert into product values(2,"banana","BusyHands");
insert into product values(3,"gizmo","GizmoWorks");
insert into product values(4,"gadget","BusyHands");
insert into product values(5,"powerGizmo","PowerWorks");
Aggregate + Join Example

```
SELECT x.manufacturer, count(*)
FROM Product x, Purchase y
WHERE x.pname = y.product
GROUP BY x.manufacturer
```

What do these query mean?

```
SELECT x.manufacturer, y.month, count(*)
FROM Product x, Purchase y
WHERE x.pname = y.product
GROUP BY x.manufacturer, y.month
```
General form of Grouping and Aggregation

\[
\begin{align*}
\text{SELECT} & \quad S \\
\text{FROM} & \quad R_1, \ldots, R_n \\
\text{WHERE} & \quad C_1 \\
\text{GROUP BY} & \quad a_1, \ldots, a_k \\
\text{HAVING} & \quad C_2
\end{align*}
\]

Why?

- \( S \) = may contain attributes \( a_1, \ldots, a_k \) and/or any aggregates but NO OTHER ATTRIBUTES
- \( C_1 \) = is any condition on the attributes in \( R_1, \ldots, R_n \)
- \( C_2 \) = is any condition on aggregate expressions and on attributes \( a_1, \ldots, a_k \)
Evaluation steps:
1. Evaluate FROM-WHERE using Nested Loop Semantics
2. Group by the attributes $a_1, \ldots, a_k$
3. Apply condition $C_2$ to each group (may have aggregates)
4. Compute aggregates in $S$ and return the result
Empty Groups

• In the result of a group by query, there is one row per group in the result
• No group can be empty!
• In particular, count(*) is never 0

```
SELECT x.manufacturer, count(*)
FROM Product x, Purchase y
WHERE x.pname = y.product
GROUP BY x.manufacturer
```
Empty Groups: Example

\[
\text{SELECT product, count(*)} \\
\text{FROM purchase} \\
\text{GROUP BY product}
\]

5 groups in our example dataset

\[
\text{SELECT product, count(*)} \\
\text{FROM purchase} \\
\text{WHERE price > 2.0} \\
\text{GROUP BY product}
\]

3 groups in our example dataset
Empty Group Problem

What if there are no purchases for a manufacturer

SELECT x.manufacturer, count(*)
FROM Product x, Purchase y
WHERE x.pname = y.product
GROUP BY x.manufacturer
Empty Group Solution: Outer Join

```
SELECT x.manufacturer, count(y.pid)
FROM Product x LEFT OUTER JOIN Purchase y
ON x.pname = y.product
GROUP BY x.manufacturer
```