Introduction to Data Management  
CSE 344  

Lectures 4 and 5: Aggregates in SQL

Outline  
• Nulls (6.1.6 - 6.1.7)  
• Outer joins (6.3.8)  
• Aggregations (6.4.3 – 6.4.6)  
• Examples, examples, examples…

NULLS in SQL  
• Whenever we don’t have a value, we can put a NULL  
• Can mean many things:  
  – Value does not exist  
  – 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

Expected behavior:  
SELECT *  
FROM Person  
WHERE (age < 25) AND (height > 6 OR weight > 190)

Rule in SQL: include only tuples that yield TRUE

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:
– \( x \text{ IS NULL} \)
– \( x \text{ IS NOT NULL} \)

Now it includes all Person tuples

Outerjoins

\[
\text{SELECT Product.name, Purchase.store}
\text{FROM Product, Purchase}
\text{WHERE Product.name = Purchase.prodName}
\]

But Products that never sold will be lost!

Outerjoins

\[
\text{SELECT Product.name, Purchase.store}
\text{FROM Product}
\text{LEFT OUTER JOIN Purchase}
\text{ON Product.name = Purchase.prodName}
\]

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
create table Purchase
(pid int primary key, product varchar(15), price float, quantity int, month varchar(15));
.import data.txt Purchase
```
Simple Aggregations

Five basic aggregate operations in SQL
• select count(*) from Purchase
• select count(quantity) from Purchase
• select sum(quantity) from Purchase
• select avg(price) from Purchase
• select max(quantity) from Purchase
• select min(quantity) from Purchase

Except count, all aggregations apply to a single attribute

Aggregates and NULL Values

Null values are not used in aggregates
• insert into Purchase values(11, 'gadget', NULL, NULL, 'april')

Let's try the following:
• select count(*) from Purchase
• select count(quantity) from Purchase
• select sum(quantity) from Purchase

COUNT applies to duplicates, unless otherwise stated:

```sql
SELECT Count(product)
FROM Purchase
WHERE price > 4.99
``` same as Count(*)

We probably want:

```sql
SELECT Count(DISTINCT product)
FROM Purchase
WHERE price > 4.99
``` Counting Duplicates

More Examples

```sql
SELECT Sum(price * quantity)
FROM Purchase
``` What do they mean?

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

Grouping and Aggregation

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

```sql
SELECT product, Sum(quantity) AS TotalSales
FROM Purchase
WHERE price > 1
GROUP BY product
``` Let's see what this means...

<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 Sum(price * quantity)
FROM Purchase
WHERE product = 'Bagel'

90 (= 60+30)
Grouping and Aggregation

1. Compute the FROM and WHERE clauses.
2. Group by the attributes in the GROUPBY clause.
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>

**Other Examples**

- `SELECT product, count(*) AS SumQuantity, max(price) AS MaxPrice FROM Purchase GROUP BY product`
- `SELECT product, count(*) AS SumQuantity, max(price) AS MaxPrice FROM Purchase GROUP BY product`
- `SELECT month, count(*) FROM Purchase GROUP BY month`

What does it mean?

### Need to be Careful…

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

SQLite is **WRONG** on this query.

- SQL Server correctly 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(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 contributed 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

Aggregates and Joins

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

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

What do these query mean?

```sql
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

```sql
SELECT S FROM R_1, ..., R_n WHERE C_1 GROUP BY a_1, ..., a_k HAVING C_2
```

S = may contain attributes a_1, ..., a_k and/or any aggregates but NO OTHER ATTRIBUTES
C_1 = is any condition on the attributes in R_1, ..., R_n
C_2 = is any condition on aggregate expressions and on attributes a_1, ..., a_k

Semantics

Evaluation steps:
1. Evaluate FROM-WHERE, apply condition C_1
2. Group by the attributes a_1, ..., 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

```
SELECT product, count(*)
FROM purchase
GROUP BY product
```

```
SELECT product, count(*)
FROM purchase
WHERE price > 2.0
GROUP BY product
```

4 groups in our example dataset

3 groups in our example dataset

Empty Group Problem

```
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
```