Introduction to Database Systems
CSE 414
Lecture 5: SQL Aggregates

Joins in SQL

- The join we have just seen is sometimes called an **inner join**
  - Each row in the result **must come from both tables in the join**
- Sometimes we want to include rows from only one of the two tables: **outer join**

(Inner) Joins

\[
\begin{align*}
\text{SELECT } & x_1.a_1, x_2.a_2, \ldots, x_m.a_m \\
\text{FROM } & \text{R}_1 \text{ as } x_1, \text{R}_2 \text{ as } x_2, \ldots, \text{R}_m \text{ as } x_m \\
\text{WHERE } & \text{Cond}
\end{align*}
\]

for \(x_1\) in \(R_1\):
for \(x_2\) in \(R_2\):
... for \(x_m\) in \(R_m\):
  if \(\text{Cond}(x_1, x_2, \ldots)\):
    \text{output}(x_1.a_1, x_2.a_2, \ldots, x_m.a_m)

This is called nested loop semantics since we are interpreting what a join means using a nested loop.

Self Join Example

- **Product**\((\text{pname, price, category, manufacturer})\)
- **Company**\((\text{cname, country})\)

Find US companies that manufacture both 'gadgets' and 'photo' products

\[
\begin{align*}
\text{SELECT DISTINCT } & z.cname \\
\text{FROM } & \text{Product } x, \text{Product } y, \text{Company } z \\
\text{WHERE } & z.country = \text{"USA"} \\
& x.manufacturer = z.cname \\
& y.manufacturer = z.cname \\
& x.category = \text{"gadget"} \\
& y.category = \text{"photography"} \\
\end{align*}
\]

Need to include Product twice!

Joins in SQL

- The join we have just seen is sometimes called an **inner join**
  - Each row in the result **must come from both tables in the join**
- Sometimes we want to include rows from only one of the two tables: **outer join**
Outer Joins

- **Left outer join:**
  - Include tuples from tableA even if no match
- **Right outer join:**
  - Include tuples from tableB even if no match
- **Full outer join:**
  - Include tuples from both even if no match
- In all cases:
  - Patch tuples without matches using NULL

```
SELECT tableA (LEFT/RIGHT/FULL) OUTER JOIN tableB ON p
```

Aggregates in SQL

```
SELECT count(*) FROM Purchase
SELECT sum(quantity) FROM Purchase
SELECT avg(price) FROM Purchase
SELECT max(quantity) FROM Purchase
SELECT min(quantity) FROM Purchase
```

Simple Aggregations

**Five basic aggregate operations in SQL**

Except count, all aggregations apply to a single attribute

```
Demo
```

```
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
```

```
COUNT applies to duplicates, unless otherwise stated:
```

```
SELECT count(product) FROM Purchase
```

same as count(*) if no nulls

```
COUNT applies to duplicates, unless otherwise stated:
```

```
SELECT count(DISTINCT product) FROM Purchase
```

We probably want:

```
SELECT count(DISTINCT product) FROM Purchase
WHERE price > 4.99
```

```
Counting Duplicates
```

```
CSE 414 - Spring 2018
7
```

```
CSE 414 - Spring 2018
8
```

```
CSE 414 - Spring 2018
9
```

```
CSE 414 - Spring 2018
10
```

```
CSE 414 - Spring 2018
11
```

```
CSE 414 - Spring 2018
12
```
More Examples

```sql
SELECT Sum(P.price * P.quantity)
FROM Purchase as P
```

```sql
SELECT Sum(P.price * P.quantity)
FROM Purchase as P
WHERE P.product = 'bagel'
```

What do they mean?

Grouping and Aggregation

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

```
SELECT product, Sum(quantity) AS TotalSales
FROM Purchase
GROUP BY product
```

Other Examples

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

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

Need to be Careful…

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

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

What does it return?

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	0.5	50
Banana	2	10
Banana	4	10

Product	Price	Quantity
Bagel	3	20
Bagel	1.50	20
Banana	0.5	50
Banana	2	10
Banana	4	10
Everything in SELECT must be either a GROUP-BY attribute, or an aggregate.

**Grouping and Aggregation**

**Purchase(product, price, quantity)**

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

\[
\text{SELECT } \text{product, Sum(quantity) AS TotalSales}
\]\n
\text{FROM Purchase}

\text{WHERE price > 1}

\text{GROUP BY product}

How is this query processed?
Grouping and Aggregation

**Purchase**(product, price, quantity)

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
```

Do these queries return the same number of rows? Why?

```sql
SELECT product, Sum(quantity) AS TotalSales FROM Purchase GROUP BY product
```

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**

<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>

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

---

**3,4: Grouping, 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>

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

---

**Ordering Results**

```sql
SELECT product, sum(price*quantity) as rev FROM Purchase GROUP BY product ORDER BY rev desc
```

Note: some SQL engines want you to say `ORDER BY sum(price*quantity) desc`

---

**HAVING Clause**

Same query as before, 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.
General form of Grouping and Aggregation

\[
\text{SELECT } S \\
\text{FROM } R_{1},...,R_{n} \\
\text{WHERE } C_1 \\
\text{GROUP BY } a_{1},...,a_{k} \\
\text{HAVING } C_2
\]

- \( S \) 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 \( a_1, \ldots, a_k \)

Why?

Semantics of SQL With Group-By

\[
\text{SELECT } S \\
\text{FROM } R_{1},...,R_{n} \\
\text{WHERE } C_1 \\
\text{GROUP BY } a_{1},...,a_{k} \\
\text{HAVING } C_2
\]

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