Ordering Results

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

```sql
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.
Exercise

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”
Exercise

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
SELECT month, sum(price*quantity), sum(quantity) as TotalSold
```
Exercise

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
SELECT    month, sum(price*quantity),
          sum(quantity) as TotalSold
FROM      Purchase
```
Exercise

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```sql
SELECT month, sum(price*quantity),
       sum(quantity) as TotalSold
FROM Purchase
GROUP BY month
```
Exercise

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
SELECT month, sum(price*quantity), sum(quantity) as TotalSold
FROM Purchase
GROUP BY month
HAVING sum(quantity) < 10
```
Exercise

Compute the total income per month
Show only months with less than 10 items sold
Order by quantity sold and display as “TotalSold”

```
SELECT month, sum(price*quantity), sum(quantity) as TotalSold
FROM Purchase
GROUP BY month
HAVING sum(quantity) < 10
ORDER BY sum(quantity)
```
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
Aggregate + Join Example

What do these queries mean?

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

```
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

<table>
<thead>
<tr>
<th>SELECT</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM</td>
<td>R_1, ..., R_n</td>
</tr>
<tr>
<td>WHERE</td>
<td>C_1</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>a_1, ..., a_k</td>
</tr>
<tr>
<td>HAVING</td>
<td>C_2</td>
</tr>
</tbody>
</table>

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

Why ?
Semantics of SQL With Group-By

```
SELECT  S  
FROM    R_1,...,R_n  
WHERE   C1  
GROUP BY a_1,...,a_k  
HAVING  C2  
```

Evaluation steps:
1. Evaluate FROM-WHERE using Nested Loop Semantics
2. Group by the attributes a_1,...,a_k
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result
Semantics of SQL With Group-By

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, \text{count(*)} is never 0

\text{SELECT} \ x.\text{manufacturer}, \ \text{count(*)} \\
\text{FROM} \ \text{Product} \ x, \ \text{Purchase} \ y \\
\text{WHERE} \ x.\text{pname} = y.\text{product} \\
\text{GROUP BY} \ x.\text{manufacturer}

What if there are no purchases for a manufacturer
Empty Group Solution: Outer Join

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