Ordering Results

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

```
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”

```sql
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”

```
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"

```sql
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

```
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

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

S = may contain attributes $a_1, ..., a_k$ and/or any aggregates but NO OTHER ATTRIBUTES

C1 = is any condition on the attributes in $R_1, ..., R_n$

C2 = is any condition on aggregate expressions and on attributes $a_1, ..., a_k$
Semantics of SQL With Group-By

```
SELECT S
FROM R₁,…,Rₙ
WHERE C₁
GROUP BY a₁,…,aₖ
HAVING C₂
```

Evaluation steps:
1. Evaluate FROM-WHERE using Nested Loop Semantics
2. Group by the attributes a₁,…,aₖ
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

```
SELECT S
FROM R_1, \ldots, R_n
WHERE C_1
GROUP BY a_1, \ldots, a_k
HAVING C_2
```
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 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
```