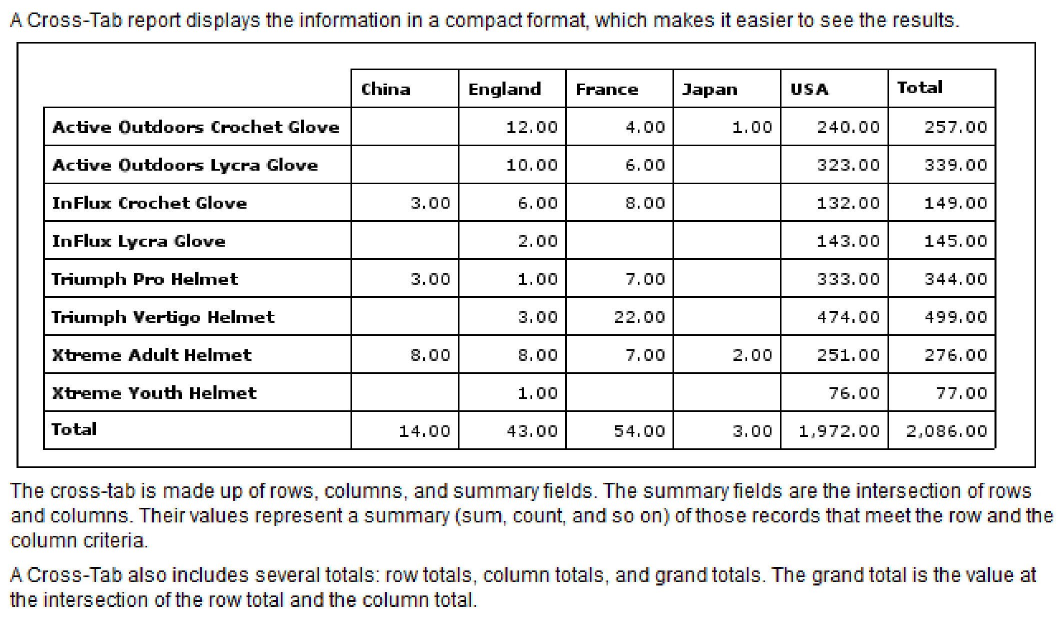
Table of Contents

This is one of those topics where people tell me they spent/wasted hours (and hours) looking this up on the internet and got more confused. So don't do that yet. One of the problems is that a CrossTab result is a generic report type of result and there is more than one way to do this. Some dbms has developed their own proprietary techniques for this process. Some web pages take you into rollups - which we do later. Some web pages go into correlated subqueries which we have not covered yet.

This document discusses one technique which should work across all dbms commonly used. That makes it valuable for people who work in more than one system. And this technique is required for the assignment. In a job situation (or on the final exam) you should use whichever technique is most efficient for your situation. But for now we are sticking with creating a cross tab report using the case expression and grouping.

1. What is a cross tab report and who wants one

If you do a search for "cross tab report" and look at images, you should get an idea that these are very common business reports. This is the way that a lot of people need to see their data organized. These are a few screen shots.



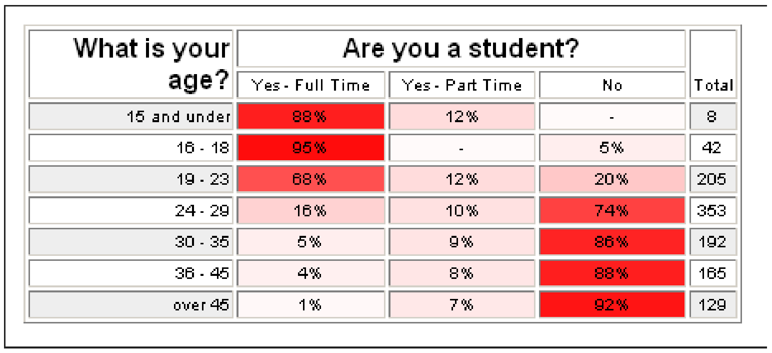
A Cross-Tab report displays the information in a compact format, which makes it easier to see the results.

	China	England	France	Japan	USA	Total
Active Outdoors Crochet Glove		12.00	4.00	1.00	240.00	257.00
Active Outdoors Lycra Glove		10.00	6.00		323.00	339.00
InFlux Crochet Glove	3.00	6.00	8.00		132.00	149.00
InFlux Lycra Glove		2.00			143.00	145.00
Triumph Pro Helmet	3.00	1.00	7.00		333.00	344.00
Triumph Vertigo Helmet		3.00	22.00		474.00	499.00
Xtreme Adult Helmet	8.00	8.00	7.00	2.00	251.00	276.00
Xtreme Youth Helmet		1.00			76.00	77.00
Total	14.00	43.00	54.00	3.00	1,972.00	2,086.00

The cross-tab is made up of rows, columns, and summary fields. The summary fields are the intersection of rows and columns. Their values represent a summary (sum, count, and so on) of those records that meet the row and the column criteria.

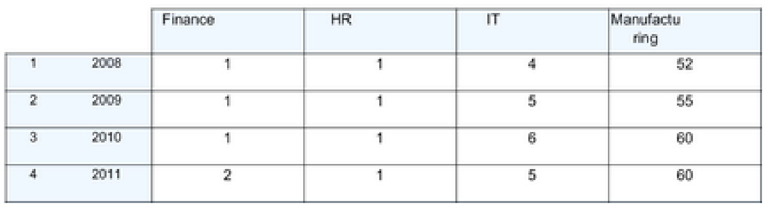
A Cross-Tab also includes several totals: row totals, column totals, and grand totals. The grand total is the value at the intersection of the row total and the column total.

http://www-01.ibm.com/support/knowledgecenter/SS4JCV_7.5.5/com.businessobjects.integration.eclipse.designer.doc/html/topic299.html



What is your age?	Are you a student?			
	Yes - Full Time	Yes - Part Time	No	Total
15 and under	88%	12%	-	8
16 - 18	95%	-	5%	42
19 - 23	68%	12%	20%	205
24 - 29	16%	10%	74%	353
30 - 35	5%	9%	86%	192
36 - 45	4%	8%	88%	165
over 45	1%	7%	92%	129

Department Expense Reports Filed



		Finance	HR	IT	Manufactu ring
1	2008	1	1	4	52
2	2009	1	1	5	55
3	2010	1	1	6	60
4	2011	2	1	5	60

A cross tab reports looks more like a spreadsheet display. In these examples we have three sources of data. In the Expense Reports Filed example we have years(2008,2009, 2010,2011). These come down the first column in the result. We have Departments (Finance, HR, IT, Manufacturing). These go across the first row of the result. The third type of data is the number (count) of reports files by each department for each year. These go into the cells.

A cross tab query normally aggregates data and displays it with the **aggregate** (sum or count or others) values as the cells and two sets of grouping data- one across the top and the other down the left side.

Some of the examples include totals as a right-most column- the first example has a total for all countries as the right most column. We will do that in this discussion. Some of the examples have a total down the columns as the last row in the display. We do that with roll up and we won't discuss that in this unit.

When you look at images of crosstab reports you will also see color shading of data, you may see more than one data value in a cell. These features are generally added by report writing applications.

This is a simpler example that we could do with our vets data. This is a simple grouping query with the total fees by animal type for exams in 2103; this is not a cross tab query. This is not using this semester's data- this is just a sample display.

```
AN_TYPE                    FEETOTAL
-------------------------- --------
cat                          899.46
porcupine                    299.50
lizard                       515.00
hedgehog                     110.00
dog                          201.00
chelonian                    200.00
dormouse                     275.00
```

But we want to break the fees down by quarters.

```
AN_TYPE                    QUARTER  FEETOTAL
-------------------------- ------- --------
cat                        2          899.46
chelonian                  1          100.00
chelonian                  2          100.00
```

```
dog                 4        201.00
dormouse            4        275.00
hedgehog            4        110.00
lizard              1        125.00
lizard              2        145.00
lizard              3         50.00
lizard              4        195.00
porcupine           1         49.50
porcupine           3        200.50
porcupine           4         49.50
```

That really doesn't look very good. It is harder to compare quarter by quarter and we do not have any rows for quarters with no fees.

The following is easier to read and it is easy to add a column for the total year and rows for animal types with no exams if we want that. This is a cross tab query. The column at the left shows the an_type values; the header row displays the second grouping condition- the quarter of the year in which the exam took place. It looks more like a standard business report.

The three data sources are: animal type, the exam date(specifically the quarter) and the **total** fees for each quarter for each animal type. We do have a Total (all_2013) column on the right.

```
AN_TYPE      QRT1_2013   QRT2_2013   QRT3_2013   QRT4_2013 ALL_2013
---------- ----------- ----------- ----------- ----------- --------
bird              0.00        0.00        0.00        0.00     0.00
cat               0.00      899.46        0.00        0.00   899.46
chelonian       100.00      100.00        0.00        0.00   200.00
dog               0.00        0.00        0.00      201.00   201.00
dormouse          0.00        0.00        0.00      275.00   275.00
hamster           0.00        0.00        0.00        0.00     0.00
hedgehog          0.00        0.00        0.00      110.00   110.00
lizard          125.00      145.00       50.00      195.00   515.00
porcupine        49.50        0.00      200.50       49.50   299.50
snake             0.00        0.00        0.00        0.00     0.00
```

We could also transpose this and list the quarters in the first column and the an_types across the first row.

One reason to refer to this as a report, rather than a query result is that the order of the columns and rows is significant to the usability of the report.

2. Using Aggregates & Case for a Cross Tab

We will use the Product.products table and create a cross tab query for the total quantity sold for selected category of item for each order. The category grouping will be the row header and the order_id grouping will be the first column. The three sources of data will be order_id (in the first column), certain product categories as the first row and the total quantity (sum) as the data points.

We can start by looking at ways to get the total quantity sold for the HD category- hardware as in hammers.

This is where we are heading in the first set of demos: What is the total quantity sold for each of these product categories?

```
+------------+-------------+------------+-------------+-------------+
| HD_QtySold | SPG_QtySold | HW_QtySold | GFD_QtySold | APL_QtySold |
+------------+-------------+------------+-------------+-------------+
|         29 |         305 |        200 |           0 |          84 |
+------------+-------------+------------+-------------+-------------+
```

Demo 01: First we can display the current data we have for HD orders. I picked a category with few sales so that you can double check the calculations

```
select order_id, OD.prod_id, cast(OH.order_date as date) as order_date,
OD.quantity_ordered as HD_QuantitySold
```

```
from OrderEntry.orderHeaders OH
join OrderEntry.orderDetails OD using(order_id)
join Product.products PR using(prod_id)
where PR.catg_id = 'HD';
+----------+---------+------------+-----------------+
| order_id | prod_id | order_date | HD_QuantitySold |
+----------+---------+------------+-----------------+
|      400 |    5002 | 2015-10-15 |               5 |
|      401 |    5002 | 2015-10-15 |               3 |
|      402 |    5002 | 2015-10-18 |               3 |
|      400 |    5004 | 2015-10-15 |               5 |
|      400 |    5005 | 2015-10-15 |               5 |
|      407 |    5005 | 2015-11-15 |               1 |
|      407 |    5005 | 2015-11-15 |               1 |
|      400 |    5008 | 2015-10-15 |               5 |
|      407 |    5008 | 2015-11-15 |               1 |
+----------+---------+------------+-----------------+
```

Demo 02: If we filter for a value of catg_id, we get an aggregate across the table and one row returned. That is the total quantity ordered for hardware items.
I do not need the order headers table for this query.

```
select sum(quantity_ordered) as HD_QuantitySold
from OrderEntry.orderDetails OD
join Product.products PR using(prod_id)
where PR.catg_id = 'HD';
+-----------------+
| HD_QuantitySold |
+-----------------+
|              29 |
+-----------------+
```

Demo 03: We can also write a query for the number of details lines that were for a HD item.

```
select count(order_id) as HD_Detail_lines
from OrderEntry.orderDetails OD
join Product.products PR using(prod_id)
where PR.catg_id = 'HD';
+-----------------+
| HD_Detail_lines |
+-----------------+
|               9 |
+-----------------+
```

Demo 04: We also can find out how many orders included a HD item. Be certain you understand the difference between this query and the previous one- both in terms of the syntax and in terms of what the task is asking for.

```
select count(distinct order_id) as HD_Orders
from OrderEntry.orderDetails OD
join Product.products PR using(prod_id)
where PR.catg_id = 'HD';
+-----------+
| HD_Orders |
+-----------+
|         4 |
+-----------+
```

Now we are going to work on getting closer to the cross tab based on the sum of the quantity.

Demo 05: We can also use the case expression to include only the rows for HD in the total. This means we do not need the Where clause filter. If the item is included in the HD category we include it in the sum and if not , we don't.

```
select
    sum(case when catg_id = 'HD' then quantity_ordered else null end) as
HD_QuantitySold
from OrderEntry.orderDetails  OD
join Product.products PR using(prod_id);
+-----------------+
| HD_QuantitySold |
+-----------------+
|              29 |
+-----------------+
```

Demo 06: You create a case expression for **each** column that you want returned. The first column does the sum for the hardware; if the row is for a hardware item(HD), then its quantity is part of the Sum for that column. The second column does the sum for the sporting goods items.

```
select
  sum(case when catg_id = 'HD'  then quantity_ordered else null end) as HD_QtySold
, sum(case when catg_id = 'SPG' then quantity_ordered else null end) as SPG_QtySold
, sum(case when catg_id = 'HW'  then quantity_ordered else null end) as HW_QtySold
, sum(case when catg_id = 'GFD' then quantity_ordered else null end) as GFD_QtySold
, sum(case when catg_id = 'APL' then quantity_ordered else null end) as APL_QtySold
from OrderEntry.orderDetails
join Product.products using(prod_id);
+------------+-------------+------------+-------------+-------------+
| HD_QtySold | SPG_QtySold | HW_QtySold | GFD_QtySold | APL_QtySold |
+------------+-------------+------------+-------------+-------------+
|         29 |         305 |        200 |        NULL |          84 |
+------------+-------------+------------+-------------+-------------+
```

A few things to note here. (1) Each column case test is set for a specific category and the column alias is hard coded for that category. (2) We have chosen 5 categories only. there are other categories but we are not concerned with them. (3) We got a null for the GFD category.

Why did we get a null for GFD? Take that first query and change it to filter for GFD ; we get no rows returned. We do not have any orders for that category.

Now we need to think about that for awhile. What should we display if there are no orders for a category? This actually is a business rule level decision- but what are our possibilities? We tend to think of NULL as meaning we don't have the data. In this case we know that with our query a Null means there are no orders for that category. We can then say that we do know what that means- the GFD total quantity sold is 0.You need to take a lot of care with a decision like that. (It does not means that all numeric nulls should be represented as 0. It means that in this situation, that decision would make sense.)

Demo 07: We could handle this by wrapping a coalesce around EACH of the sum expressions

```
select coalesce(sum(case when catg_id = 'HD'
               then quantity_ordered else null end),0) as HD_QtySold
. . . .
from OrderEntry.orderDetails
join Product.products using(prod_id);
```

Demo 08: We could also- in this case, rewrite the case expression to return 0 if the category id is not matched. The Sum aggregate function ignores null, but arithmetically we can add 0 to a running total (SUM) without changing the total. (This will not work the same way with Avg, Max, Min.)

```
/*  Demo 08  */
```

```
select sum(case when catg_id = 'HD'  then quantity_ordered else 0 end) as
HD_QtySold
   . . . .
   from OrderEntry.orderDetails
   join Product.products using(prod_id);
```

2.1. Adding a grouping

So far we did not display a leading column - we have only two data sources- the category and the total quantity. We will add the order id as another data source and display this in the first column. To do that we need to group on the order id to get one row per order id.

Demo 09: We want to know how many products of each of these categories are on EACH order so we add a grouping on the order id. Because I do not have a lot of data in the tables, we get a lot of 0 values. I have selected rows that show the HD sales.

```
select
  order_id
, sum(case when catg_id = 'HD'  then quantity_ordered else 0 end) as HD_QtySold
, sum(case when catg_id = 'SPG' then quantity_ordered else 0 end) as SPG_QtySold
, sum(case when catg_id = 'HW'  then quantity_ordered else 0 end) as HW_QtySold
, sum(case when catg_id = 'GFD' then quantity_ordered else 0 end) as GFD_QtySold
, sum(case when catg_id = 'APL' then quantity_ordered else 0 end) as APL_QtySold
from OrderEntry.orderDetails
join Product.products using(prod_id)
group by order_id;
+--------+------------+-------------+------------+-------------+-------------+
| ord_id | HD_QtySold | SPG_QtySold | HW_QtySold | GFD_QtySold | APL_QtySold |
+--------+------------+-------------+------------+-------------+-------------+
|    390 |          0 |           8 |          0 |           0 |           0 |
|    395 |          0 |          15 |          0 |           0 |           0 |
|    400 |         20 |           0 |          0 |           0 |           0 |
|    401 |          3 |           0 |          0 |           0 |           0 |
|    402 |          3 |           0 |          0 |           0 |           0 |
|    405 |          0 |           6 |          0 |           0 |           0 |
|    407 |          3 |           0 |          0 |           0 |           0 |
|    408 |          0 |           0 |          1 |           0 |           0 |
|    411 |          0 |           2 |          4 |           0 |           0 |
|    412 |          0 |           0 |          0 |           0 |           1 |
```

You could write a version that produces spaces instead of 0's- that results in this case in a lot of spaces and make it harder to read across the report. The sql is in the demo.

2.1. Doing a count instead of a sum

Now let's count the number of orders for HD products instead of getting the total quantity.

You might just take a previous demo and change the SUM to COUNT, but that could pose problems.

Demo 10: Suppose you did the following expression for each category (see the demo file for the query)

```
select  count(case when catg_id = 'HD'  then quantity_ordered else 0 end) as HD_QtySold
. . .
```

The result would be the following . WHY?

```
+--------+---------+--------+---------+---------+
| HD_??? | SPG_??? | HW_??? | GFD_????| APL_??? |
+--------+---------+--------+---------+---------+
|    184 |     184 |    184 |     184 |     184 |
+--------+---------+--------+---------+---------+
```

And counting the quantity_ordered column should seem wrong. If you want to find out how many Orders- count the order_id. That gives us two more versions. They are in the demo file, but try to figure these out for yourself first. If all you do is look at demos, you do not learn much.

Demo 11: Getting a count

```
+------------+------------+------------+------------+------------+
| HD_Orders  | SPG_Orders | HW_Orders  | GFD_Orders | APL_Orders |
+------------+------------+------------+------------+------------+
|          9 |         49 |         54 |          0 |         27 |
+------------+------------+------------+------------+------------+
```

Demo 12: Getting a different count

```
+------------+------------+------------+------------+------------+
| HD_Orders  | SPG_Orders | HW_Orders  | GFD_Orders | APL_Orders |
+------------+------------+------------+------------+------------+
|          4 |         30 |         39 |          0 |         23 |
+------------+------------+------------+------------+------------+
```

2.2. Filtering the entire data source and using a case expression

A lot of business reports want sales data organized by time periods. You can do this by using a date function in the case expression.

Demo 13: How many orders for each customer for each of these three months of last year?
The only table needed is the order headers table. In this case I can count a literal instead of the order id.

```
select customer_id
,  count(case when month(order_date) = 10 then 1 end) as "Oct"
,  count(case when month(order_date) = 11 then 1 end) as "Nov"
,  count(case when month(order_date) = 12 then 1 end) as "Dec"
from OrderEntry.orderHeaders
where year(order_date)= year( current_date) - 1
group by customer_id
order by customer_id
;
+-------------+-----+-----+-----+
| customer_id | Oct | Nov | Dec |
+-------------+-----+-----+-----+
|      400300 |   0 |   0 |   0 |
|      401250 |   1 |   2 |   0 |
|      401890 |   0 |   1 |   0 |
|      402100 |   0 |   3 |   0 |
|      403000 |   3 |   1 |   0 |
|      403010 |   0 |   1 |   0 |
|      403050 |   1 |   0 |   0 |
|      403100 |   3 |   1 |   0 |
```

Suppose we want to display the data by quarter for the year 2015 and include a total column.
First we join the tables, using an outer join to include products with no sales

```
from Product.products   PR
left join OrderEntry.orderDtails OD  on OD.prod_id = PR.prod_id
left join OrderEntry.orderHeaders OH on OH.order_id = OD.order_id
```

Then we filter for the year- we want sales for the year 2015. If there are no sales the order date would be null and we want to include those also.

```
where year(order_date)= 2015 or order_date is null
```

We want to data organized by the category id

```
   group by catg_id
```

Now we can come back to the display- the Select. We have 6 columns- the first is the catgory id, the next column is the total sales for the first quarter, followed by columns the total sales for the second quarter etc and a total column.

We are using the case expression to determine the quarter for each order date; the mysql has a quarter function

```
   case when quarter(order_date) =1
```

If the sales is in the quarter we want we get the extended cost, otherwise treat it as 0.

```
   case when quarter(order_date) =1
          then quantity_ordered * quoted_price else 0 end) as Quart1_2015
```

We want to total of all of the sales for that quarter, so we need the sum aggregate function.

We then add a last column that does not test the quarter

Demo 14: Display of sales by quarter for different product categories.

```
   select
   catg_id
   , sum( case when quarter(order_date) =1 then quantity_ordered * quoted_price else 0 end)
   as Qrt1_2015
   , sum( case when quarter(order_date) =2 then quantity_ordered * quoted_price else 0 end)
   as Qrt2_2015
   , sum( case when quarter(order_date) =3 then quantity_ordered * quoted_price else 0 end)
   as Qrt3_2015
   , sum( case when quarter(order_date) =4 then quantity_ordered * quoted_price else 0 end)
   as Qrt4_2015
   , coalesce(sum( quantity_ordered * quoted_price),0) as All_qrts
   from Product.products PR
   left join OrderEntry.orderDetails OD  using(prod_id)
   left join OrderEntry.orderHeaders OH using(order_id)
   where year(order_date)= 2015 or order_date is null
   group by catg_id;
   +---------+-----------+-----------+-----------+-----------+----------+
   | catg_id | Qrt1_2015 | Qrt2_2015 | Qrt3_2015 | Qrt4_2015 | All_qrts |
   +---------+-----------+-----------+-----------+-----------+----------+
   | APL     |      0.00 |  10000.00 |   4005.39 |   5724.98 | 19730.37 |
   | GFD     |      0.00 |      0.00 |      0.00 |      0.00 |     0.00 |
   | HD      |      0.00 |      0.00 |      0.00 |    695.65 |   695.65 |
   | HW      |      0.00 |    594.99 |   1945.90 |   1309.93 |  3850.82 |
   | MUS     |      0.00 |      0.00 |    314.60 |      0.00 |   314.60 |
   | PET     |      0.00 |    774.25 |    494.85 |   1194.12 |  2463.22 |
   | SPG     |      0.00 |  14079.95 |  16550.25 |   3069.00 | 33699.20 |
   +---------+-----------+-----------+-----------+-----------+----------+
   7 rows in set (0.03 sec)
```

This gives us 7 rows. But we have 9 different product categories. Why did we not get 9 rows?

See the alternate versions of the query in the demo file.

Demo 15: This might not be considered a cross tab query because it does not have two grouping levels, but it is similar in using the Case technique.
Analyze quantity of items purchased by price

```
   select
      sum(case when quoted_price between 0.01 and 25
          then quantity_ordered
          else 0 end)  as "Price 0.01-25"
   ,  sum(case when quoted_price between 25.01 and 100
          then quantity_ordered
          else 0 end)  as "Price 25.01-100"
```

```
,  sum(case when quoted_price between 100.01 and 250
        then quantity_ordered
        else 0 end)  as "Price 100.01- 250"
,  sum(case when quoted_price >  250
        then quantity_ordered
        else 0 end)  as "Price > 250"
,  sum(quantity_ordered) as "Tot Quant"
from OrderEntry.orderDetails;
+---------------+-----------------+-------------------+-------------+-----------+
| Price 0.01-25 | Price 25.01-100 | Price 100.01- 250 | Price > 250 | Tot Quant |
+---------------+-----------------+-------------------+-------------+-----------+
|           506 |             115 |               240 |          89 |       950 |
+---------------+-----------------+-------------------+-------------+-----------+
```

Demo 16: A different layout for this query result. The expressions in the Select and the Group By are identical.

```
select
case
  when quoted_price between 0.01 and 25    then 'Price   0.01 -  25'
  when quoted_price between 25.01 and 100  then 'Price  25.01 - 100'
  when quoted_price between 100.01 and 250 then 'Price 100.01 - 250'
  when quoted_price >  250                 then 'Price over 250'
end as "Price Range"
,
sum(quantity_ordered) AS "Total Quantity"
from  OrderEntry.orderDetails
group by case
      when quoted_price between 0.01 and 25    then 'Price   0.01 -  25'
      when quoted_price between 25.01 and 100  then 'Price  25.01 - 100'
      when quoted_price between 100.01 and 250 then 'Price 100.01 - 250'
      when quoted_price >  250                 then 'Price over 250'
      end
order by "Price Range"

+--------------------+----------------+
| Price Range        | Total Quantity |
+--------------------+----------------+
| Price   0.01 -  25 |            506 |
| Price  25.01 - 100 |            115 |
| Price 100.01 - 250 |            240 |
| Price over 250     |             89 |
+--------------------+----------------+
```

Demo 17: a different version of the query using an in-line view.

```
select PriceRange as "Price Range"
, select PriceRange as "Price Range"
, sum(quantity_ordered) as "Total Quantity"
from ( select
     case
       when quoted_price between 0.01 and 25    then 'Price   0.01 -  25'
       when quoted_price between 25.01 and 100  then 'Price  25.01 - 100'
       when quoted_price between 100.01 and 250 then 'Price 100.01 - 250'
       when quoted_price >  250                 then 'Price over 250'
     end as PriceRange
   , quantity_ordered
   from OrderEntry.orderDetails
)SalesAnalysis
group by  PriceRange
order by  PriceRange;
```