CHAPTER

2

What's in a Table? Getting Started with Data Exploration

The previous chapter introduced the SQL language from the perspective of data analysis. This chapter uses SQL for exploring data, the first step in any analysis project. The emphasis shifts away from databases in general. Understanding what the data represents—and the underlying customers—is a theme common to this chapter and the rest of the book.

The most common data analysis tool, by far, is the spreadsheet, particularly Microsoft Excel. Spreadsheets show data in a tabular format. They give users power over the data, with the ability to add columns and rows, to apply functions, to summarize, create charts, make pivot tables, and color and highlight and change fonts to get just the right look. This functionality and the what-you-see-is-what-you-get interface make spreadsheets a natural choice for analysis and presentation.

Spreadsheets, however, are less powerful than databases because they are designed for interactive use. The historical limits in Excel on the number of rows (once upon a time, a maximum of 65,535 rows) and the number of columns (once upon a time, a maximum of 255 columns) clearly limited the spreadsheets to smaller applications. Even without those limits, spreadsheet applications often run on a local machine and are best applied to single tables (workbooks). They are not designed for combining data stored in disparate formats. The power of users' local machines can limit the performance of spreadsheet applications.

This book assumes a basic understanding of Excel, particularly familiarity with the row-column-worksheet format used for laying out data. There are many examples using Excel for basic calculations and charting. Because charts are so important for communicating results, the chapter starts by reviewing some of the charting tools in Excel, providing tips for creating good charts.

The chapter continues with exploring data in a single table, column by column. Such exploration depends on the types of data in the column, with separate sections devoted to numeric columns and categorical columns. Although dates and times are touched upon here, they are so important that Chapter 4 is devoted to them. The chapter ends with a method for automating some descriptive statistics for columns in general. Most of the examples in this chapter use the purchases dataset, which describes retail purchases.

What Is Data Exploration?

Data is stored in databases as bits and bytes, spread through tables and columns, in memory and on disk. Data lands there through various business processes. Operational databases capture the data as it is collected from customers—as they make airplane reservations, or complete telephone calls, or click on the web, or as their bills are generated. The databases used for data analysis are usually decision support databases and data warehouses where the data has been restructured and cleansed to conform to some view of the business.

Data exploration is the process of characterizing the data actually present in a database and understanding the relationships between various columns and entities. Data exploration is a hands-on effort. Metadata, documentation that explains what *should* be there, provides one description. Data exploration is about understanding what actually *is* there, and, if possible, understanding how and why it got there. Data exploration is about answering questions about the data:

- What are the values in each column?
- What unexpected values are in each column?
- Are there any data format irregularities, such as time stamps missing hours and minutes, or names being both upper- and lowercase?
- What relationships are there between columns?
- What are frequencies of values in columns and do these frequencies make sense?

TIP **Documentation tells us what should be in the data; data exploration finds what is actually there.**

Almost anyone who has worked with data has stories about data quality or about discovering something very unexpected inside a database. At one telecommunications company, the billing system maintained customers'

telephone numbers as an important field inside the data. Not only was this column stored as character strings rather than numbers, but several thousand telephone *numbers* actually contained *letters* intermixed with numbers. Clearly, the column called "telephone number" was not always a telephone number. And, in fact, after much investigation, it turned out that under some circumstances (involving calls billed to third parties), the column could contain values other than numbers.

Even when you are familiar with the data, exploration is still worthwhile. The simplest approach is just to look at rows and sample values in tables. Summary tables provide a different type of information. Statistical measures are useful for characterizing data. Charts are very important because a good chart can convey much more information than a table of numbers. The next section starts with this topic: charting in Excel.

Excel for Charting

Excel's charting capabilities give users much control over the visual presentation of data. A good presentation of results, however, is more than just clicking an icon and inserting a chart. Charts need to be accurate and informative, as well as visually elegant and convincing. Edward Tufte's books, starting with *The Visual Display of Quantitative Information*, are classics in how to display and convey information.

This section discusses various common chart types and good practices when using them. The discussion is necessarily specific, so some parts explain explicitly, click-by-click, what to do. The section starts with a basic example and then progresses to recommended formatting options. The intention is to motivate good practices by explaining the reasons, not to be a comprehensive resource explaining, click-by-click, what to do in Excel.

A Basic Chart: Column Charts

The first example, in Figure 2-1, uses a simple aggregation query, the number of orders for each payment type. The chart format used is a column chart, which shows a value for each column. In common language, these are also called bar charts, but in Excel, bar charts have horizontal bars whereas column charts have vertical columns.

The query that pulls the data is:

```
SELECT PaymentType, COUNT(*) as cnt
FROM Orders o
GROUP BY PaymentType
ORDER BY PaymentType
```

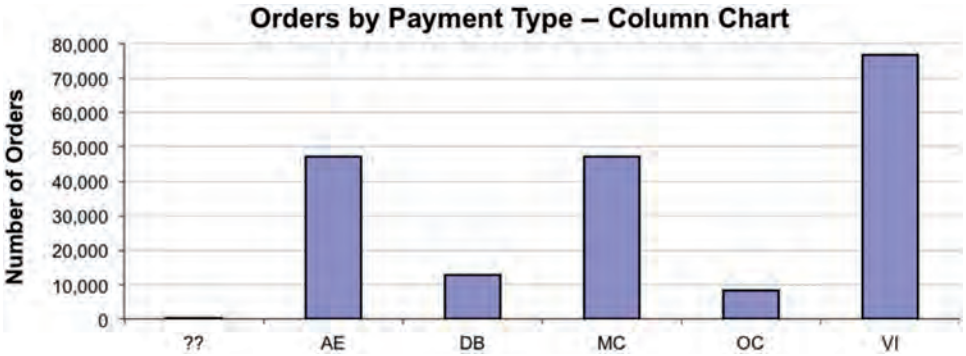


Figure 2-1: A basic column chart shows the number of orders for each payment type code.

This chart shows some good practices:

- The chart has a title.
- Appropriate axes have labels (none is needed for the horizontal axis because its meaning is clear from the title).
- Numbers larger than one thousand have commas because people are going to read the values.
- Horizontal gridlines are useful, but they are light so they do not overpower the data.
- Extraneous elements are kept to a minimum. For instance, there is no need for a legend (because there is only one series) and no need for vertical grid lines (because the columns serve the same purpose).

For the most part, charts throughout the book adhere to these conventions, with the exception of the title. Figures in the book have captions, making titles unnecessary. The rest of this section explains how to create the chart with these elements.

Inserting the Data

Creating the chart starts with running the query and getting the data into an Excel spreadsheet. The data is assumed to be generated by a database access tool, which can copy data into Excel using cut-and-paste (Ctrl+C and Ctrl+V, if the tool conforms to Windows standards, or Command+C and Command+V on a Mac) or other methods. The previous query produces two columns of data. It is also possible to run SQL directly from Excel by setting up a data source. Although useful for automated reports, such data connections are less useful for data exploration efforts using many ad hoc queries.

A good practice is to include the query in the spreadsheet along with the data itself. Including the query above the data ensures that you know how the data was generated, even when you return to it hours, days, or months after running the query.

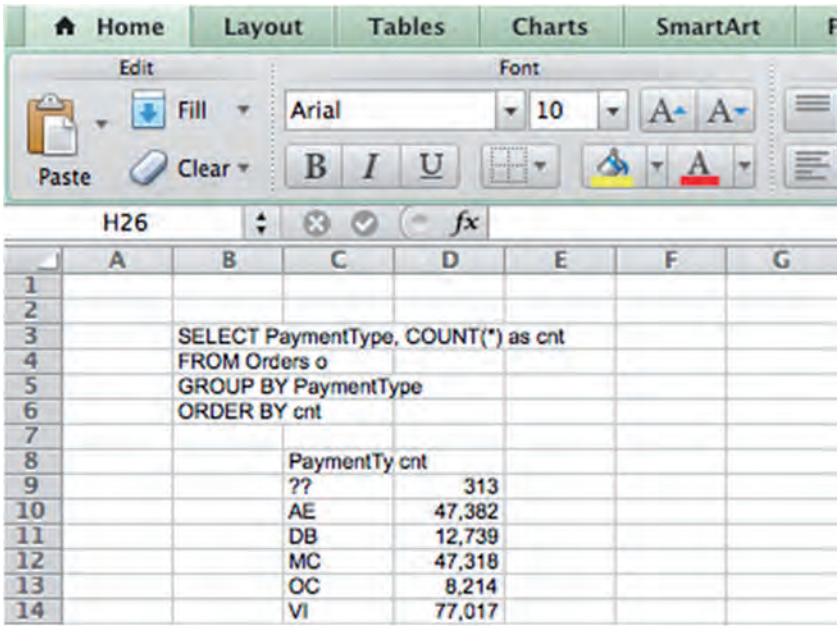


Figure 2-2: This spreadsheet contains the column data for payment types and orders.

TIP **Keeping the query with the results is always a good idea. So, copy the query into the Excel spreadsheets along with the data.**

The technical aside "Common Issues When Copying Data into Excel" discusses some issues that occur when copying data. In the end, the spreadsheet looks something like Figure 2-2. Notice that this data includes the query used to generate the data.

Creating the Column Chart

Creating a column chart—or any other type of chart—has just two considerations. The first is inserting the chart; the second is customizing it to be clean and informative.

The simplest way to create the chart is with the following steps:

1. Highlight the data that goes into the chart. In this case, the query results have two columns and both columns, the payment type code and the count (along with their headers), go into the chart. If there is a non-data line between the header and the data, delete it (or copy the headers into the cells just above the data). To use keystrokes instead of the mouse, go to the first cell and type Shift+Ctrl+<down arrow> (or Shift+Command+<down arrow> on a Mac).
2. Bring up the Chart wizard. Use the Charts ribbon to select the Column chart, which is the first option.

COMMON ISSUES WHEN COPYING DATA INTO EXCEL

Each database access tool has its own peculiarities when copying data into Excel. One method is to export the data as a file and import the file into Excel. One issue when copying the data directly from the clipboard is the data landing in a single column. The second is a lack of headers in the data. A third issue is the formatting of the columns themselves.

Under some circumstances, Excel places copied data in a single column rather than in multiple columns. This problem occurs because Excel recognizes the values as text rather than as columns.

This problem is easily solved by converting the text to columns:

1. Highlight the inserted data that you want converted to columns. Use either the mouse or keystrokes. For keystrokes, go to the first cell and type Shift+Ctrl+<down arrow> (Command+Shift+<down arrow> on a Mac).
2. Bring up the "Text to Columns" wizard by going to the Data ribbon and choosing the Text to Columns tool. (This tool can also be accessed from the Data menu).
3. Choose appropriate options. The data may be delimited by tabs or commas, or each column may have a fixed width. Buttons at the top of the wizard let you choose the appropriate format.
4. Finish the wizard. Usually the remaining choices are unimportant. The one exception is when you want to import columns that look like numbers but are not. To keep leading zeros or minus signs, set the column data format to text.
5. When finished, the data is transformed into columns, filling the columns to the right of the original data.

The second problem is a lack of headers. Older versions of SQL Server Management Studio, for instance, did not offer an easy way to copy headers. In these versions, you can set up SQL Server Management Studio to copy the headers along with the data by going to Tools ➢ Options ➢ Query Results ➢ SQL Server ➢ Results to Grid and checking "Include column headers when copying or saving the results."

The third issue is the formatting of columns. Column formats are important; people read the data and formats help convey the meaning: $10,011 is very different from the zip code 10011.

By default, large numbers do not have commas. One way to insert commas is to highlight the column, right click, and choose "Format." Go to the "Number" tab, choose "Number," set "0" decimal places, and click the "Use 1000 Separator" box. Date fields usually need to have their format changed. For them, go to the "Custom" option and type in the string **yyyy-mm-dd**. This sets the date format to a standard format. To set dollar amounts, choose the "Currency" option, with "2" as the decimal places and "$" (or the appropriate character) as the symbol.

3. Choose the first option, "Clustered Column" and the chart appears.
4. To add a title, go to the Chart Format ribbon and select Chart Title ➢ Title Above Chart. Triple-click on the text box that appears to select all the text and type **Number of Orders by Payment Type**.
5. To set the Y-axis, choose Axis Titles ➢ Vertical Axis Title ➢ Rotated Title, triple-click in the box (to highlight the current value) and type **Num Orders**.
6. Resize the chart to an appropriate size, if you like.

A chart, formatted with the default options, now appears in the spreadsheet. This chart can be copied and pasted into other applications, such as PowerPoint, Word, and email applications. When pasting the chart into other applications, it can be convenient to paste the chart as a picture rather than as a live Excel chart. To do this, use the File ➢ Paste Special menu option and choose the picture option.

Formatting the Column Chart

The following are the formatting conventions to apply to the column chart:

- Resize the chart in the chart window
- Format the legend
- Change the fonts
- Change border
- Adjust the horizontal scale

For reference, Figure 2-3 shows the names of various components of a chart, such as the *chart area*, *plot area*, *horizontal gridlines*, *chart title*, *X-axis label*, *Y-axis label*, *X-axis title*, and *Y-axis title*.

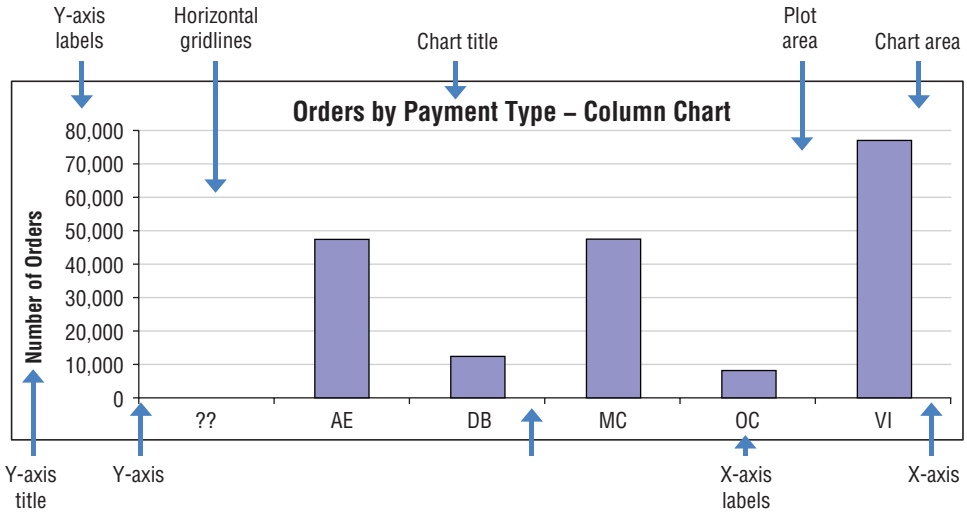


Figure 2-3: An Excel chart consists of many different parts.

Resizing the Chart in the Chart Window

By default, the chart does not take up quite all the space in the chart window. Why waste space? Click the gray area to select the plot area. Then make it bigger, keeping in mind that you usually don't want to cover the chart title and axis labels.

Formatting the Legend

By default, Excel adds a legend, containing the name of each series in the chart. Having a legend is a good thing. By default, though, the legend is placed next to the chart, taking up a lot of real estate and shrinking the plot area. In most cases, it is better to have the legend overlap the plot area. To do this, click the plot area (the actual graphic in the chart window) and expand to fill the chart area. Then, click the legend and move it to the appropriate place, somewhere where it does not cover data values.

When there is only one series, a legend is unnecessary. To remove it, just click the legend box and hit the Delete key.

Changing the Fonts

To change all the fonts in the chart at once, double-click the white area to select options for the entire chart window. On the "Font" tab, deselect "Auto scale" on the lower left. Sizes and choices of fonts are definitely a matter of preference, but 8-point Arial is a reasonable choice.

This change affects all fonts in the window. The chart title should be larger and darker (such as Arial 12-point Bold), and the axis titles a bit larger and darker (such as Arial 10-point Bold). You can just click on the chart title and change the font on the Home ribbon.

Changing the Border

To remove the outer border on the entire plot area, double-click the white space to bring up the "Format Chart Area" dialog box. Choose the "Line" option and set the Color to "None."

Adjusting the Grid Lines

Grid lines should be visible to make chart values more readable. However, the grid lines are merely sideshows on the chart; they should be faint, so they do not interfere with or dominate the data points. On column charts, only horizontal grid lines are needed; these make it possible to easily match the vertical scale to the data points. On other charts, both horizontal and vertical grid lines are recommended.

By default, Excel includes the horizontal grid lines but not the vertical ones. To choose zero, one, or both sets of grid lines, go to the Chart Layout ribbon and use the "Gridlines" option. The "Major Gridlines" boxes for both the X and Y axes are useful. The "Minor Gridlines" are rarely needed. You can also adjust the

colors by using the "Gridlines Options …" on the same menu. A good choice of colors is the lightest shade of gray, just above the white. Note that you can also right-click on the gridlines in the chart to bring up similar menus.

Adjust the Horizontal Scale

For a column chart, every category should be visible. By default, Excel might only show some of the category names. To change this, double-click the horizontal axis to bring up the "Format Axis" dialog box, and go to the "Scale" tab. Set the second and third numbers, "Number of categories between tick-mark labels" and "Number of categories between tick-marks" both to 1. This controls the spacing of the marks on the axis and of the labels. Note that you can also get to this menu using the "Axes" option on the Chart Layout ribbon.

TIP **To include text in a chart that is connected to a cell (and whose value changes when the cell value changes), insert a text box into the chart. Then select the text box, type the equals character (=), and click the cell with the value you want. A text box appears with the text; this can be formatted and moved however you choose. The same technique works for other text boxes, such as titles. On a Mac, you can do something similar, but you need to insert a smart shape (using Insert ➢ Picture ➢ Shape) and then assign it to a cell the same method.**

Bar Charts in Cells

Excel charts are powerful, but sometimes they are overkill for conveying simple information. Excel also offers methods for putting charts directly in cells. The simplest is a bar chart, where a single bar is located inside a cell, instead of a value. There are two approaches to creating such "in-cell" charts. The first is more brute-force, based on character strings, and the second uses conditional formatting.

Character-Based Bar Charts

Repeating single characters makes a passable bar chart, as shown in Figure 2-4. The power of such a chart is that it shows the data and the relative values of the data at the same time. The bars clearly show that MC and AE are basically equal in popularity and VI is the most popular.

The "chart" is really just a string created with the Excel function `REPT()`. This function takes a character and repeats it:

- `REPT("|", 3)` ➢ |||
- `REPT("-", 5)` ➢ -----

The repetition of the character looks like a bar chart. Vertical bars and dashes are useful characters for this purpose.

AE	47,382	----------
DB	12,739	---
MC	47,318	----------
OC	8,214	--
VI	77,017	----------
AE	47,382	
DB	12,739	
MC	47,318	
OC	8,214	
VI	77,017	
AE	47382	=REPT("g", 20*D9/MAX($D$8:$D$13))
DB	12739	=REPT("g", 20*D10/MAX($D$8:$D$13))
MC	47318	=REPT("g", 20*D11/MAX($D$8:$D$13))
OC	8214	=REPT("g", 20*D12/MAX($D$8:$D$13))
VI	77017	=REPT("g", 20*D13/MAX($D$8:$D$13))

Figure 2-4: Bar charts can be created within a cell using variable strings of characters.

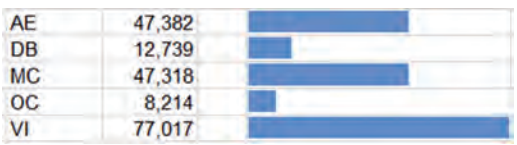
A trick produces the nicer middle chart in Figure 2-4. The trick is to use the lowercase "g" and convert the font to Webdings. In this font, the lowercase "g" is a filled-in square that forms a nice bar.

The lower part of Figure 2-4 shows the formulas that are used for this chart. There is nothing special about "20" in the formulas; that is simply the maximum length of the bars.

Conditional Formatting-Based Bar Charts

Bar charts within a cell is so useful that Excel actually builds in this functionality. On the Home ribbon, the conditional formatting option is under "Format" (you can also access it from the Menu option Format ➢ Conditional Formatting). Under this menu is an option for "Data Bars."

Figure 2-5 shows what happens when you choose this option. The length of the bars is automatically determined, so no additional calculations are needed. There is a problem: The values in the cells overlap the bars. The solution is to make the values disappear by using the format specification. The ideal format specification would be an empty string for any value, but this is not allowed. Instead, go to the Number Format menu, choose "Custom," and then type in one, two, or three semicolons.



AE	47,382
DB	12,739
MC	47,318
OC	8,214
VI	77,017

Figure 2-5: Data bars can be produced using conditional formatting.

Why does this work? The format specification for a cell can have different formats for positive, negative, zero values, and text values. Semicolons separate the different formats for each of these possible values. By having nothing between the semicolons, the value is not displayed at all. The bars from conditional formatting are still displayed.

TIP **Format specifications are quite powerful. They are even powerful enough to prevent values from being displayed in a cell (which is convenient when using conditional formatting to color the cell or create a bar within the cell).**

Useful Variations on the Column Chart

This simple column chart illustrates many of the basic principles of using charts in Excel. To illustrate some useful variations, a somewhat richer set of data is needed.

A New Query

A richer set of data provides more information about the payment types, information such as:

- Number of orders with each code
- Number of orders whose price is in the range $0–$10, $10–$100, $100–$1,000, and over $1,000
- Total revenue for each code

The following query uses conditional aggregation to calculate these results:

```
SELECT PaymentType,
       SUM(CASE WHEN 0 <= TotalPrice AND TotalPrice < 10
                THEN 1 ELSE 0 END) as cnt_0_10,
       SUM(CASE WHEN 10 <= TotalPrice AND TotalPrice < 100
                THEN 1 ELSE 0 END) as cnt_10_100,
       SUM(CASE WHEN 100 <= TotalPrice AND TotalPrice < 1000
                THEN 1 ELSE 0 END) as cnt_100_1000,
       SUM(CASE WHEN TotalPrice >= 1000 THEN 1 ELSE 0 END) as cnt_1000,
       COUNT(*) as cnt, SUM(TotalPrice) as revenue
FROM Orders
GROUP BY PaymentType
ORDER BY PaymentType
```

The data divides the orders into four groups, based on the size of the orders. This is a good set of data for showing different ways to compare values using column charts.

Side-by-Side Columns

Side-by-side columns, as shown in the top chart in Figure 2-6, are the first method for the comparison. This chart shows the actual value of the number

of orders for different groups. Some combinations are so small that the column is not even visible.

This chart clearly illustrates two points. First, three payment methods predominate: AE (American Express), MC (MasterCard), and VI (Visa). Second, orders in the range of $10 to $100 predominate.

To create such a side-by-side chart, choose the "Clustered Column" chart with multiple columns selected.

Stacked Columns

The middle chart in Figure 2-6 shows stacked columns. This communicates the total number of orders for each payment type, making it possible to find out, for instance, where the most popular payment mechanisms are. Stacked columns maintain the actual values; however, they do a poor job of communicating proportions, particularly for smaller groups.

To create stacked columns, choose the "Stacked Columns" chart option.

Stacked and Normalized Columns

Stacked and normalized columns provide the ability to see proportions across different groups, as shown in the bottom chart in Figure 2-6. Their drawback is that small numbers—in this case, very rare payment types—have as much weight visually as the more common ones. These outliers can dominate the chart.

One solution is to include payment type codes that have only some minimum number of orders. Filtering the data, by going to the Data ribbon and choosing filter (or using the Data ➢ Filter ➢ Autofilter menu option), is one way to do this. Another is by sorting the data in descending order by the total count, and then choosing the top rows to include in the chart.

To create the chart, choose the "100% Stacked Columns" chart.

Number of Orders and Revenue

Figure 2-7 shows another variation, where one column has the number of orders, and the other has the total revenue. The number of orders varies up to several tens of thousands. The revenue varies up to several millions of dollars. On a chart with both series, the number of orders would disappear because the numbers are so much smaller.

The trick is to plot the two series using different scales, which in Excel lingo means plotting them on different axes. This chart has the number of orders on the left and the total revenue on the right. Set the colors of the axes and axis labels to match the colors of the columns.

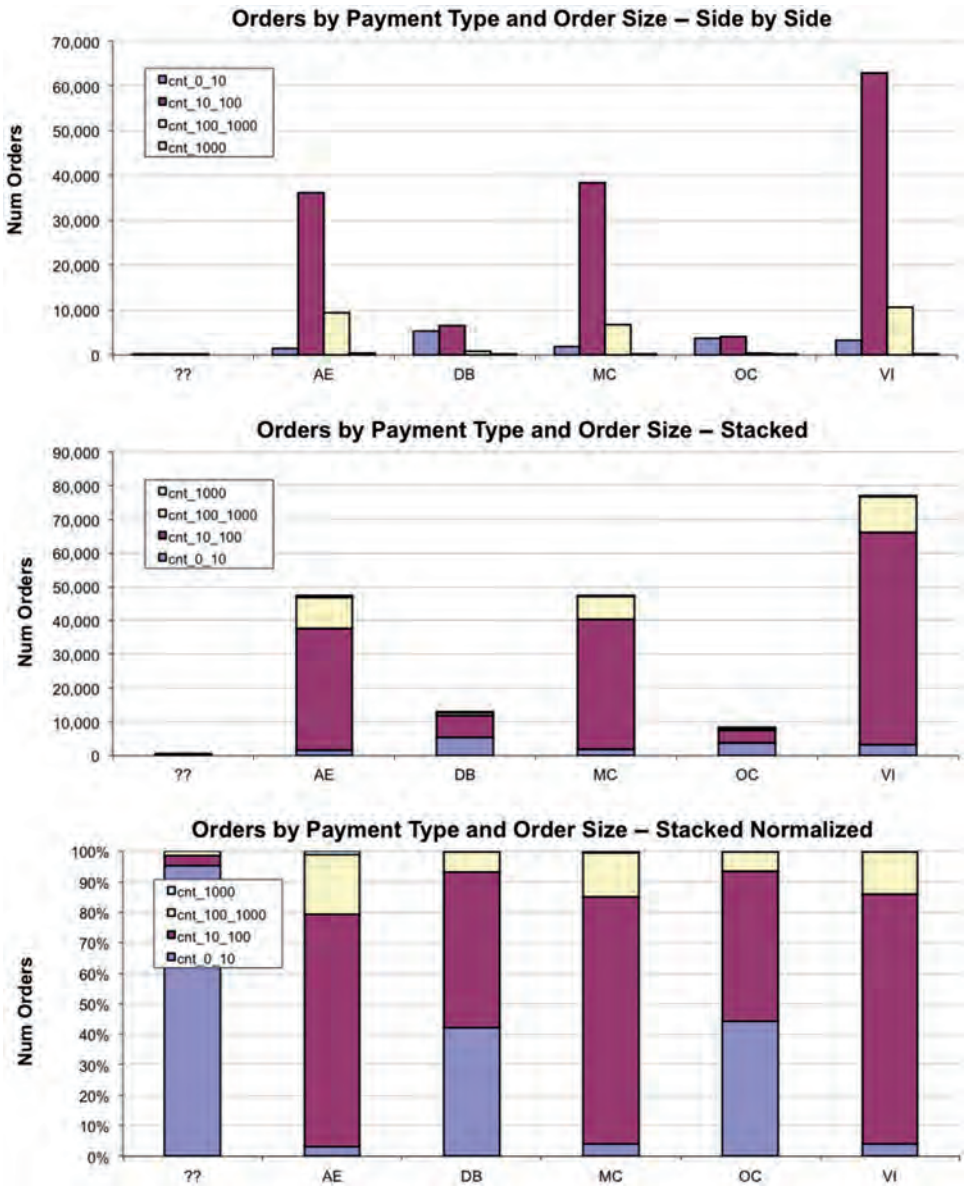


Figure 2-6: Three different charts using the same data emphasize different types of information, even though they contain the same raw data.

Using a second axis for column charts creates overlapping columns. To get around this, the columns for the number of orders are wide and those for the revenue are narrow. Also, either chart can be modified to be of a different type, making it possible to create many different effects.

To make such a chart, first include revenue and number of orders data in the chart, by selecting all the data and then removing (or unselecting) the series

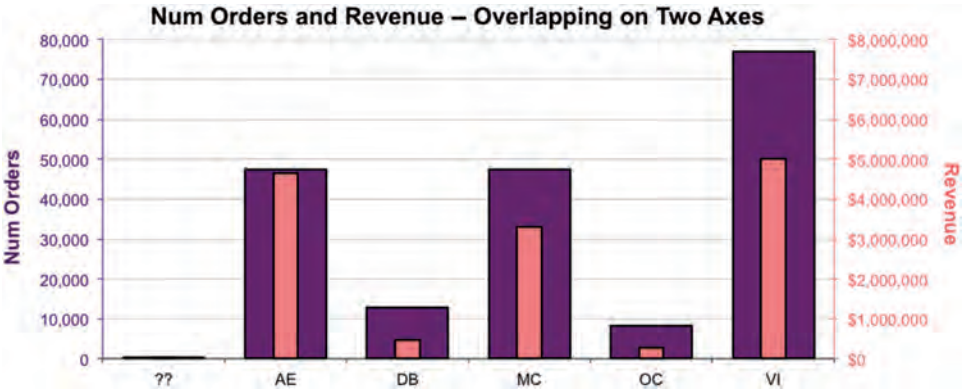


Figure 2-7: Showing the number of orders and revenue requires using two axes.

one by one. After inserting the chart, right-click and choose "Select Data" to bring up a dialog box with the series on the left. One by one, remove the series that should not be part of the chart (that is, all but the two series we want). An alternative method is to add each series separately into the chart.

Second, the revenue series needs to move to the secondary axis. Select the series either by right-clicking on it or by going to the Chart Layout ribbon and choosing the series in the Current Selection box on the far left. Choose "Format Data Series…" (if right-clicking) or "Format Selection" (if on the ribbon). Go to the "Axis" tab and click "Secondary axis."

Third, add a title to the secondary axis by going to the Chart Format ribbon and choosing "Axis Titles." The bottom choice is "Secondary Vertical Axis Title." After adding the title, change the colors of the two axes to match the series. By matching the colors, you can eliminate the legend, reducing clutter on the chart.

When creating charts with two Y-axes, the gridlines should align to the tick-marks on both axes. This typically requires manual adjustment. In this case, set the scale on the right-hand axis so the maximum is $8,000,000, instead of the default $6,000,000. To do this, double-click the axis, go to the "Scale" tab, and change the "Maximum" value. The gridlines match the scales on both sides.

TIP When creating charts with series on both axes, try to make the gridlines match up on both sides by adjusting the scales on the axes so they align.

The final step is to get the effect of the fat and skinny columns. To create the fat column, double-click the number of orders data columns. On the "Options" tab, set the "Overlap" to 0 and the "Gap Width" to 50. To get the skinny columns, double-click the revenue data series. Set the "Overlap" to 100 and the "Gap Width" to 400.

Other Types of Charts

Other types of charts are used throughout the book. This section is intended as an introduction to these charts. Many of the options are similar to the options for the column charts, so the specific details do not need to be repeated.

Line Charts

The data in the column charts can also be represented as line charts, such as in Figure 2-8. Line charts are particularly useful when the horizontal axis represents a time dimension because they naturally show changes over time. Line charts can also be stacked the same way as column charts, including normalized and stacked.

Line charts have some interesting variations that are used in later chapters. The simplest is deciding whether the line should have icons showing each point, or simply the line that connects them. Choosing the chart subtype controls this.

Line charts also have the ability to add trend lines and error bars, features that get used in later chapters.

Area Charts

Area charts show data as a shaded region. They are similar to column charts, but instead of columns, there is only the colored region with no spaces between data points. They should be used sparingly because they fill the plot area with color that does not convey much information. They are primary used for series on the secondary axis using lighter, background colors.

Figure 2-9 shows the total orders as columns (with no fill on the columns) and the total revenue presented as an area chart on the secondary Y-axis. This chart emphasizes that the three main payment types are responsible for most orders and most revenue. Notice, that AE and MC have about the same number

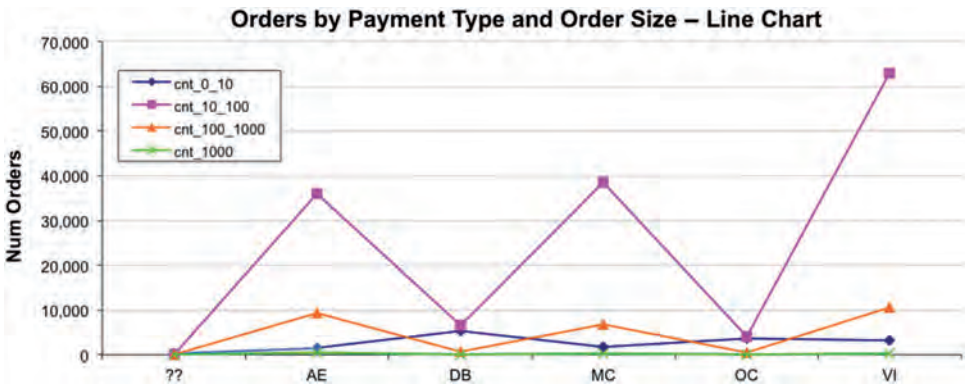


Figure 2-8: The line chart is an alternative to a column chart. Line charts can make it easier to spot certain types of trends.

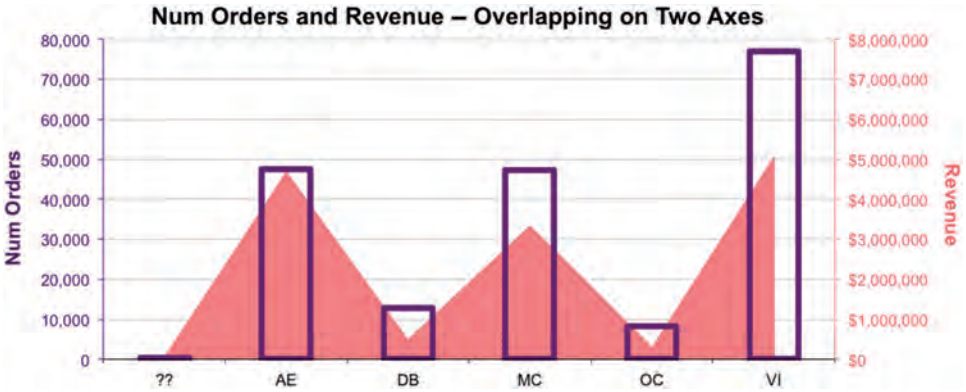


Figure 2-9: This example shows the revenue on the secondary axis as an area chart.

of orders, but AE has much more revenue. This means that the average revenue for customers who pay by American Express is larger than the average revenue for customers who pay by MasterCard.

To create this chart, follow the same steps as used for Figure 2-7. Click once on the number of orders series to choose it. Then right-click and choose "Change Series Chart Type…." Then choose "Area" on the Charts ribbon. To change the colors, double-click the colored area and choose appropriate borders and colors for the region.

To change the column fill to transparent, double-click the number of orders series, to bring up the "Format Data Series" dialog. Click on "Fill," and for "Color" choose "No Fill."

X-Y Charts (Scatter Plots)

Scatter plots are very powerful and are used for many examples. Figure 2-10 has a simple scatter plot that shows the number of orders and revenue for each

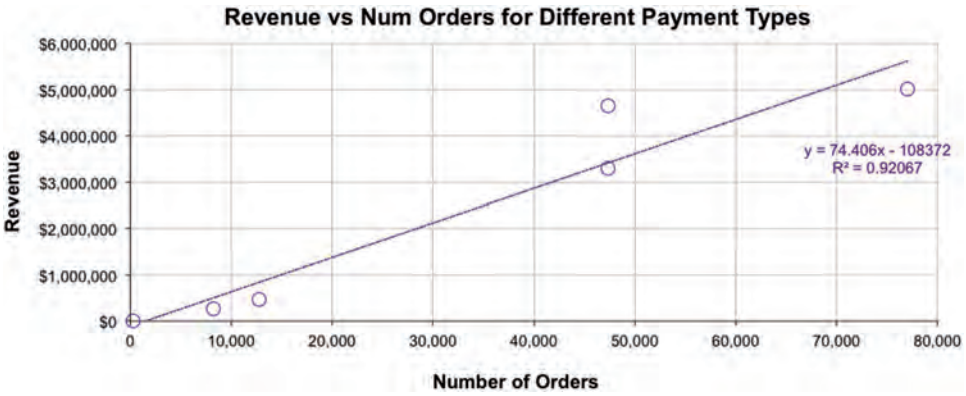


Figure 2-10: This scatter plot shows the relationship between the number of orders and revenue for various payment types.

payment type. This example has both horizontal and vertical gridlines, which is recommended for scatter plots.

Unfortunately, Excel does not allow labeling of the points on the scatter plot with codes or other information. You have to go back to the original data to see what the points refer to. The point above the trend line is for American Express—orders paid using American Express have more revenue than the trend line suggests.

This example shows an obvious relationship between the two variables—payment types with more orders have more revenue. According to the equation for the trend line, each additional order brings in about $75 additional revenue. To see the relationship, add a trend line (which is discussed in more detail in Chapter 12). Click the series to choose it, then right-click and choose "Add Trendline…." On the "Options" tab, you can choose to see the equation by clicking the button next to the "Display equation on Chart." Click "OK" and the trend line appears. It is a good idea to make the trend line a similar color to the original data, but lighter, perhaps using a dashed line. Double-clicking the line brings up a dialog box with these options.

This section has discussed credit card types without any discussion of how to determine the type. The aside "Credit Card Numbers" discusses the relationship between credit card numbers and credit card types.

Sparklines

A sparkline is a special type of chart that fits within a single cell. Typically, these are column charts or line charts that are particularly useful for showing changes over time. Excel offers many fewer options for formatting sparklines, but like in-cell bar charts, they have the tremendous advantage of being shown with the data itself.

TIP **Sparklines are particularly useful for showing trends.**

How do the number of purchases vary by month for different payment types? To answer this question, let's focus on a single year, 2015. The idea is to summarize the orders data by month and then create sparklines that show the changes through the year.

The query to extract the data uses conditional aggregation:

```
SELECT PaymentType,
       SUM(CASE WHEN MONTH(OrderDate) = 1 THEN 1 ELSE 0 END) as Jan,
        . . .
       SUM(CASE WHEN MONTH(OrderDate) = 12 THEN 1 ELSE 0 END) as Dec
FROM Orders o
WHERE YEAR(OrderDate) = 2015
GROUP BY PaymentType
ORDER BY PaymentType
```

The "…" is not part of SQL; it is shorthand for the missing ten months.

CREDIT CARD NUMBERS

This section used payment types as the example without explaining how credit card types are extracted from credit card numbers. Credit card numbers are not random; they have some structure:

- **The first six digits are the Bank Identification Number (BIN). These are a special case of Issuer Identification Numbers defined by an international standard called ISO 7812.**
- **An account number follows, controlled by whoever issues the credit card.**
- **A checksum is at the end to verify the card number is valid.**

Credit card numbers themselves are interesting, but don't use them! Storing credit card numbers, unencrypted in a database, poses privacy and security risks. However, there are two items of interest in the numbers: the credit card type and whether the same credit card is used on different transactions.

Extracting the credit card type, such as Visa, MasterCard, or American Express, is only challenging because the folks who issue the BINs are quite secretive about who issues which number. However, over the years, the most common credit card types have become known (Wikipedia is a good source of information). The BINs for the most common credit card types are in the following table:

PREFIX	CC TYPE
34, 37	AMEX
560, 561	DEBIT
300–305, 309, 36, 38, 39, 54, 55	DINERS CLUB
6011, 622126–622925, 644-649, 65	DISCOVER
2014, 2149	enRoute
3528–3589	JCB
50–55	MASTERCARD
4	VISA

The length of the prefix typically varies from one number to four numbers, which makes it a bit difficult to do a lookup in Excel. The following CASE **statement assigns credit card types in SQL:**

```
SELECT (CASE WHEN ccn LIKE '34%' OR ccn LIKE '37%'
             THEN 'AMEX'
             WHEN ccn LIKE '560%' OR ccn LIKE '561%'
             THEN 'DEBIT'
             WHEN LEFT(ccn, 3) IN ('300', '301', '302', '303', '304',
                                   '305', '309' OR
                  LEFT(cnn, 2) IN ('36', '38', '54', '44')
```

```
            THEN 'DINERS CLUB'
            WHEN ccn LIKE '6011%' OR ccn LIKE 65%' OR
                 LEFT(ccn, 3) BETWEEN '644' and '649' OR
                 LEFT(ccn, 6) BETWEEN '622126' and '622925'
            THEN 'DISCOVER'
            WHEN LEFT(ccn, 4) IN ('2014',  '2149')
            THEN 'ENROUTE'
            WHEN LEFT(ccn, 4) BETWEEN '3528' AND '3589'
            THEN 'JCB'
            WHEN LEFT(ccn, 2) BETWEEN '50' and '55'
            THEN 'MASTERCARD'
            WHEN ccn LIKE '4%'
            THEN 'VISA'
            ELSE 'OTHER'
       END) as cctypedesc
```

Note that the conditions use a combination of operators, including `LIKE`, `LEFT()`, `BETWEEN`, and `IN`.

Recognizing when the same credit card number in different transactions can be easy and dangerous or a bit harder. The simple solution is to store the credit card number in the database. This is a bad idea, for security reasons.

A better approach is to transform the number into something that doesn't look like a credit card number. One possibility is to encrypt the number (if your database supports this). In SQL Server, `CHECKSUM()` is usually good enough, although more advanced encryption functions are supported.

Figure 2-11 shows sparklines associated with this data. Three of them are totally flat. This is because the sparklines all use the same vertical scale and these do not have large enough values to show up on the lines. For the other three, we see that all increase in December, but American Express increases more than the rest.

To insert the sparklines, choose "Sparklines" from the Insert ribbon, and then select the input cells and destination. Unlike the bar charts discussed earlier, each sparkline is in its own cell. The *Sparklines* ribbon has a choice of various formats; this example uses the linear sparkline. By default, the vertical axis varies for each sparkline, which is inconvenient for seeing patterns across rows. To make the vertical axes consistent, select a group and choose the "Axis" option on the far

PaymentType	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	
??	56	6	116	6	3	8	7	2	7	4	4	6	
AE	613	421	618	399	510	656	422	487	496	581	1,355	1,514	
DB	196	110	110	67	60	109	71	80	86	79	75	183	
MC	692	504	667	419	491	744	487	447	477	623	1,265	1,466	
OC	65	45	201	230	54	108	46	33	35	60	110	137	
VI	1,095	751	926	605	719	1,004	737	716	716	1,018	1,813	2,086	

Figure 2-11: Sparklines showing the number of purchases by month.

right of the Sparkline ribbon. Under the two sets of options for the vertical axis (for both the minimum and the maximum), choose "Same for All Sparklines."

The lower part of Figure 2-11 shows similar information, using the average purchase amount per month. The query is similar:

```
SELECT PaymentType,
       AVG(CASE WHEN MONTH(OrderDate) = 1 THEN TotalPrice END) as Jan,
        . . .
       AVG(CASE WHEN MONTH(OrderDate) = 12 THEN TotalPrice END) as Dec
FROM Orders o
WHERE YEAR(OrderDate) = 2015
GROUP BY PaymentType
ORDER BY PaymentType
```

Note that the `CASE` statement has no `ELSE` clause. The default for `CASE` is `NULL` when there is no match. This works perfectly with the average function, which ignores `NULL` values.

These sparklines use the column chart. This tells a somewhat different story. First, there is much less seasonality to the larger orders. Also, AE payers have larger average order amounts than other orders. Perhaps American Express customers are wealthier than average. Alternatively, more small businesses may use American Express and their orders might, on average, be larger than other orders.

What Values Are in the Columns?

The basic charting mechanisms are a good way to see the data, but what do we want to see? The rest of this chapter discusses things of interest when exploring a single table. Although this discussion is in terms of a single table, remember that SQL makes it quite easy to join tables together to make them look like a single table—so the methods apply equally well to multiple tables.

The section starts by investigating frequencies of values, using histograms, for both categorical and numeric values. It then continues to discuss interesting measures (statistics) on columns. Finally, it shows how to gather all these statistics in one rather complex query.

Histograms

A histogram is a chart—usually a column chart—that shows the distribution of values in a column. For instance, the following query calculates the number of orders and population in each state, answering the question: *What is the distribution of orders by state and how is this related to the state's population?*

```
SELECT State, SUM(numorders) as numorders, SUM(pop) as pop
FROM ((SELECT o.State, COUNT(*) as numorders, 0 as pop
       FROM Orders o
```

```
        GROUP BY o.state
       ) UNION ALL
       (SELECT zc.stab, 0 as numorders, SUM(totpop) as pop
        FROM ZipCensus zc
        GROUP BY zc.stab
       )
      ) summary
GROUP BY State
ORDER BY numorders DESC
```

This query combines information from the `ZipCensus` and `Orders` tables. The first subquery counts the number of orders and the second calculates the population. These are combined using `UNION ALL`, to ensure that all states that occur in either table are included in the final result. Alternatively, two queries could produce two result sets that are then combined in Excel.

Figure 2-12 shows the results. Notice that the population is shown as a lighter shaded area on the secondary axis and the number of orders as a column chart. The states are ordered by the number of orders.

The chart shows several things. California, which has the largest population, is third in number of orders. Perhaps this is an opportunity for more marketing in California. At the very least, it suggests that marketing and sales efforts are focused on the northeast because New York and New Jersey have larger numbers of orders. This chart also suggests a measure of penetration in the state, the number of orders divided by the population (although a better measure might be the number of unique customers/households divided by the number of households in the state).

The resulting chart is a bit difficult to read because there are too many state abbreviations to show on the horizontal axis. It is possible to expand the horizontal axis and make the font small enough so all the abbreviations fit, just barely. This works for state abbreviations; for other variables it might be impractical, particularly if there are more than a few dozen values.

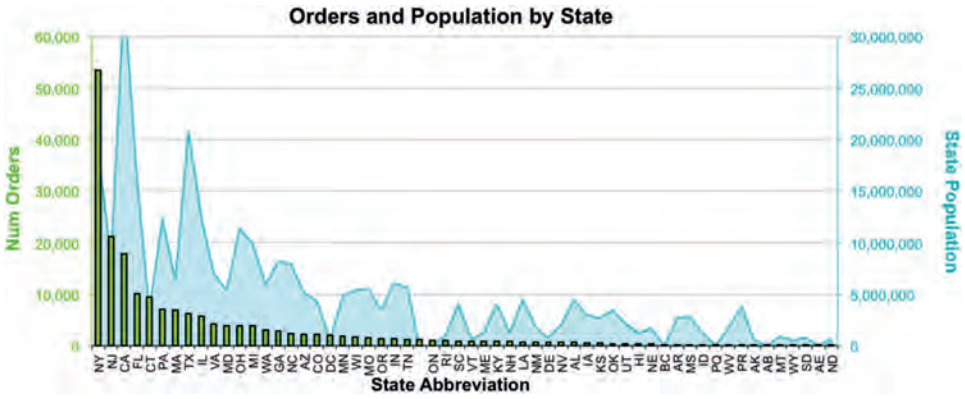


Figure 2-12: This example shows the states with the number of orders in columns and the population as an area.

One way to make the results more intelligible is to place the data into groups. That is, take the states with few orders and collect them together into one "OTHER" category; states with many orders are kept individually. Let's say that states with fewer than 100 orders are placed in the "OTHER" category. *What is the distribution of orders among states that have 100 or more orders?*

```
SELECT (CASE WHEN cnt >= 100 THEN State ELSE 'OTHER' END) as state,
       SUM(cnt) as cnt
FROM (SELECT o.State, COUNT(*) as cnt
      FROM Orders o
      GROUP BY o.State
     ) os
GROUP BY (CASE WHEN cnt >= 100 THEN state ELSE 'OTHER' END)
ORDER BY cnt DESC
```

This query puts the data in the same two-column format used previously for making a histogram. Note the use of the conditional in the `GROUP BY` column.

This approach has a drawback because it requires a fixed value in the query—the "100" in the comparison. One possible modification is to ask a slightly different question: *What is the distribution of orders by state, for states that have more than 2% of the orders?*

```
SELECT (CASE WHEN bystate.cnt >= 0.02 * total.cnt
             THEN state ELSE 'OTHER' END) as state,
       SUM(bystate.cnt) as cnt
FROM (SELECT o.State, COUNT(*) as cnt
      FROM Orders o
      GROUP BY o.State
     ) bystate CROSS JOIN
     (SELECT COUNT(*) as cnt FROM Orders) total
GROUP BY (CASE WHEN bystate.cnt >= 0.02 * total.cnt
               THEN state ELSE 'OTHER' END)
ORDER BY cnt desc
```

The first subquery calculates the total orders in each state. The second calculates the total orders. Because this subquery produces only one row, the query uses a `CROSS JOIN`. The aggregation then uses a `CASE` statement that chooses states that have at least 2% of all orders.

Actually, this query answers the question and goes one step beyond. It does not filter out the states with fewer than 2% of the orders. Instead, it groups them together into the "OTHER" group, ensuring that no orders are filtered out. Keeping all the data helps prevent mistakes in understanding the data.

Note that the "OTHER" category has changed dramatically using these two methods. In the first version, the states in the "OTHER" group are not very important. Their 422 orders put them—combined—in 41st place between Kansas and Oklahoma. The second query puts "OTHER" in second place between New York and New Jersey with 42,640 orders.

TIP **When writing exploration queries that analyze data, keeping all the data is usually a better approach than filtering rows. Use a special group to keep track of what would have been filtered.**

Another alternative is to have some number of states, such as the top 20 states, with everything else placed in the other category. *What is the distribution of the number of orders in the 20 states that have the largest number of orders?* Unfortunately, such a query is complex. The simplest approach uses a row number calculation:

```
SELECT (CASE WHEN seqnum <= 20 THEN state ELSE 'OTHER' END) as state,
       SUM(numorders) as numorders
FROM (SELECT o.State, COUNT(*) as numorders,
             ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC) as seqnum
      FROM Orders o
      GROUP BY o.State
     ) bystate
GROUP BY (CASE WHEN seqnum <= 20 THEN state ELSE 'OTHER' END)
ORDER BY numorders DESC
```

This is an example of using `ROW_NUMBER()` in an aggregation query.

This query could also be accomplished in SQL Server using the `TOP` option (other databases typically use `LIMIT` or the ANSI standard `FETCH FIRST <X> ROWS ONLY`):

```
SELECT TOP 20 o.State, COUNT(*) as numorders
FROM Orders o
GROUP BY o.State
ORDER BY COUNT(*) DESC
```

In this version, the subquery sorts the data by the number of orders in descending order. The `TOP` option then chooses the first 20 rows and returns only these. This method does not generate the "OTHER" category, so the results do not include data for all states.

An interesting variation on the histogram is the cumulative histogram, which makes it possible to calculate, for instance, how many states account for half the orders. You can add the cumulative sum to one of the above queries—for instance:

```
SELECT TOP 20 o.State, COUNT(*) as numorders,
       SUM(COUNT(*)) OVER (ORDER BY COUNT(*) DESC) as cumesum
FROM Orders o
GROUP BY o.State
ORDER BY COUNT(*) DESC
```

Note that some databases that support window functions do not support cumulative sums. Notably, versions of SQL Server prior to 2012 do not have this functionality.

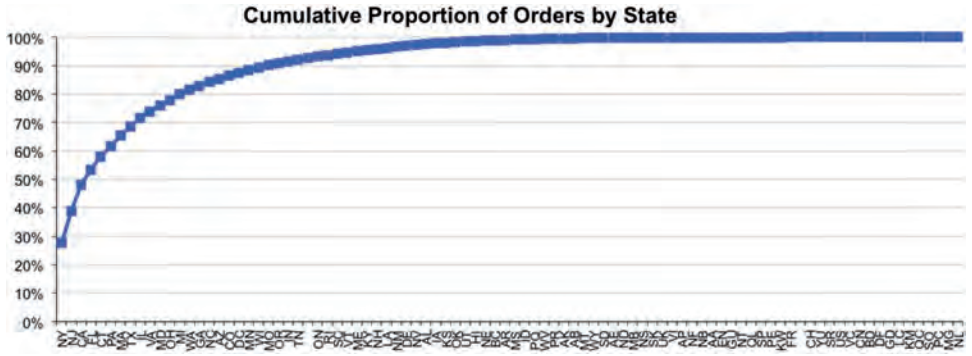


Figure 2-13: The cumulative histogram shows that four states account for more than half of all orders.

The cumulative sum can also be calculated in Excel. This process starts by ordering the results by the number of orders in descending order (so the biggest states are at the top). To add the cumulative sum, let's assume that the number of orders is in column B and the data starts in cell B2. An easy way is to type the formula `=C1+B2` in cell C2 and then copy this formula down the column. An alternative formula that does not reference the previous cell is `=SUM($B$2:$B2)`. If desired, the cumulative number can be divided by the total orders to get a percentage, as shown in Figure 2-13.

Histograms of Counts

The number of states is well known. Americans learn early that 50 states comprise the union. The Post Office recognizes 62—because places such as Puerto Rico (PR), the District of Columbia (DC), Guam (GM), and the Virgin Islands (VI) are treated as states—along with three more abbreviations for "states" used for military post offices. Corporate databases might have even more, sometimes giving equal treatment to Canadian provinces and American states, and even intermingling foreign country or province codes with state abbreviations.

Still, there are a relatively small number of states in contrast to the thousands of zip codes—more than fit in a single histogram. Where to start with such columns? A good question to ask is the histogram of counts question: *What is the number of zip codes that have a given number of orders?* The following query answers this:

```
SELECT numorders, COUNT(*) as nmzips, MIN(ZipCode), MAX(ZipCode)
FROM (SELECT o.ZipCode, COUNT(*) as numorders
      FROM Orders o
      GROUP BY o.ZipCode
     ) bystate
GROUP BY numorders
ORDER BY numorders
```

The subquery calculates the counts for each zip code. The outer SELECT counts how often each count occurs in the histogram.

The result set says how many zip codes have exactly one order, exactly two orders, and so on. For instance, in this data, 5,954 zip codes have exactly one order. The query also returns the minimum and maximum zip code values. These provide examples of zip codes with each count. The two examples in the first row are not valid zip codes, suggesting that some or all of the one-time zip codes are errors in the data.

TIP The histogram of counts for the primary key column always has exactly one row, where CNT is 1 because primary keys are never duplicated.

Another example uses OrderLines. *What is the number of order lines where the product occurs once (overall), twice, and so on?* The query that answers is also a histogram of counts:

```
SELECT numol, COUNT(*) as numprods, MIN(ProductId), MAX(ProductId)
FROM (SELECT ProductId, COUNT(*) as numol
      FROM OrderLines
      GROUP BY ProductId
     ) op
GROUP BY numol
ORDER BY numol
```

The subquery counts the number of order lines where each product appears. The outer query then creates a histogram of this number.

This query returns 385 rows; the first few rows and last row are in Table 2-1. The last row of the table has the most common product, whose ID is 12820 and

Table 2-1: Histogram of Counts of Products in OrderLines Table

NUMBER OF ORDERS	NUMBER OF PRODUCTS	MINIMUM PRODUCTID	MAXIMUM PRODUCTID
1	933	10017	14040
2	679	10028	14036
3	401	10020	14013
4	279	10025	14021
5	201	10045	13998
6	132	10014	13994
7	111	10019	13982
8	84	10011	13952
...			
18,648	1	12820	12820

appears in 18,648 order lines. The least common products are in the first row; there are 933 that occur only once—about 23.1% of all products. However, these rare products occur in only 933/286,017 orders, about 0.02% of orders.

How many different values of `ProductId` are there? This is the sum of the second column in the table, which is 4,040. How many order lines? This is the sum of the product of the first two columns, which is 286,017. The ratio of these two numbers is the average number of order lines per product, 70.8; that is, a given product occurs in 70.8 order lines, on average. The calculation in Excel uses the function `SUMPRODUCT()`, which takes two columns, multiplies them together cell by cell, and then adds the results together. The specific formula is "`=SUMPRODUCT(C13:C397, D13:D397)`."

Cumulative Histograms of Counts

What proportion of products account for half of all order lines? Answering this question requires two cumulative columns, the cumulative number of order lines and the cumulative number of products, as shown in Table 2-2.

This table shows that products with six or fewer order lines account for 65.0% of all products. However, they appear in only 2.2% of order lines. We have to go to row 332 (out of 385) to find the middle value. In this row, the product appears in 1,190 order lines and the cumulative proportion of order lines crosses the halfway point. This middle value—called the *median*—shows that 98.7% of all products account for half the order lines, so 1.3% account for the other half. In other words, the common products are much more common than the rare ones. This is an example of the long tail that occurs when working with thousands or millions of products.

Table 2-2: Histogram of Counts of Products in the OrderLines Table with Cumulative OrderLines and Products

NUMBER		CUMULATIVE		CUMULATIVE %	
ORDER LINES	PRODUCTS	ORDER LINES	PRODUCTS	ORDER LINES	PRODUCTS
1	933	933	933	0.3%	23.1%
2	679	2,291	1,612	0.8%	39.9%
3	401	3,494	2,013	1.2%	49.8%
4	279	4,610	2,292	1.6%	56.7%
5	201	5,615	2,493	2.0%	61.7%
6	132	6,407	2,625	2.2%	65.0%
...					
1,190	1	143,664	3,987	50.2%	98.7%
...					
18,648	1	286,017	4,040	100.0%	100.0%

The cumulative number of products is the sum of all values in the number of products column up to a given row. A simple formula for this calculation is `=SUM($D$284:$D284)`. When this formula is copied down the column, the first half of the range stays constant (that is, remains `$D$284`) and the second half increments (becoming `$D284` then `$D285` and so on). This form of the cumulative sum is preferable to the `=H283+D284` form because cell `H283` contains a column title, which is not a number, causing problems in the first sum. One way around this is to add `IF()` to the formula: `=IF(ISNUMBER(H283), H283, 0) + D284`.

The cumulative number of order lines is the sum of the product of the number of order lines and number of products `numol` and `numprods` values (columns C and D) up to that point. The formula is:

```
SUMPRODUCT($C$284:$C284, $D$284:$D284)
```

The ratios are the value in each cell divided by the last value in the column.

Histograms (Frequencies) for Numeric Values

Histograms work for numeric values as well as categorical ones. For instance, `NumUnits` contains the number of different units of a product included in an order and it takes on just a handful of values. How do we know this? The following query answers the question: *How many different values does* `NumUnits` *take on?*

```
SELECT COUNT(*) as numol, COUNT(DISTINCT NumUnits) as numvalues
FROM OrderLines
```

There are only 158 different values in the column. On the other hand, the column `TotalPrice` has over 4,000 values, which is a bit cumbersome for a histogram, although the cumulative histogram is still quite useful.

A natural way to look at numeric values is by grouping them into ranges. The next section explains several methods for doing this.

Ranges Based on the Number of Digits, Using Numeric Techniques

Counting the number of important digits—those to the left of the decimal point—is a good way to group numeric values into ranges. For instance, a value such as "123.45" has three digits to the left of the decimal point. For numbers greater than one, the number of digits is one plus the log in base 10 of the number, rounded down to the nearest integer:

```
SELECT FLOOR(1+ LOG(val) / LOG(10)) as numdigits
```

However, not all values are known to be greater than 1. For values between –1 and 1, the number of digits is zero, and for negative values, we might as

well identify them with a negative sign. The following expression handles these cases:

```
SELECT (CASE WHEN val >= 1 THEN FLOOR(1 + LOG(val) / LOG(10))
             WHEN -1 < val AND val < 1 THEN 0
             ELSE - FLOOR(1 + LOG(-val) / LOG(10)) END) as numdigits
```

Used in a query for `TotalPrice` in `Orders`, this turns into:

```
SELECT numdigits, COUNT(*) as numorders, MIN(TotalPrice),
       MAX(TotalPrice)
FROM (SELECT (CASE WHEN TotalPrice >= 1
                   THEN FLOOR(1 + LOG(TotalPrice) / LOG(10))
                   WHEN -1 < TotalPrice AND TotalPrice < 1 THEN 0
                   ELSE - FLOOR(1 + LOG(-TotalPrice) / LOG(10)) END
             ) as numdigits, TotalPrice
      FROM Orders o
     ) a
GROUP BY numdigits
ORDER BY numdigits
```

In this case, the number of digits is a small number between 0 and 4 because `TotalPrice` is never negative and always under $10,000. Note that the query also returns the smallest and largest values in the range—a helpful check on the values.

The following expression turns the number of digits into a lower and upper bounds, assuming that the underlying value is never negative:

```
SELECT SIGN(numdigits) * POWER(10, numdigits - 1) as lowerbound,
       POWER(10, numdigits) as upperbound
```

This expression uses the `SIGN()` function, which returns –1, 0, or 1 depending on whether the argument is less than zero, equal to zero, or greater than zero. A similar expression can be used in Excel. Table 2-3 shows the results from the query.

Table 2-3: Ranges of Values for TotalPrice in Orders Table

# DIGITS	LOWER BOUND	UPPER BOUND	# ORDERS	MINIMUM	MAXIMUM
0	$0	$1	9,130	$0.00	$0.64
1	$1	$10	6,718	$1.75	$9.99
2	$10	$100	148,121	$10.00	$99.99
3	$100	$1,000	28,055	$100.00	$1,000.00
4	$1,000	$10,000	959	$1,001.25	$9,848.96

Ranges Based on the Number of Digits, Using String Techniques

There is a small error in the table. The number "1000" is calculated to have three digits rather than four. The discrepancy is due to a rounding error in the calculation. An alternative, more exact method is to use string functions.

String functions can calculate the length of the string representing the number, using only digits to the left of the decimal place. The SQL expression for this is:

```
SELECT LEN(CAST(FLOOR(ABS(val)) as INT)) * SIGN(FLOOR(val)) as numdigits
```

This expression uses the nonstandard `LEN()` function and assumes that the integer is converted to a character value (although all databases have such a function, it is sometimes called `LENGTH()`). See Appendix A for equivalent functions in other databases.

More Refined Ranges: First Digit Plus Number of Digits

Table 2-4 shows the breakdown of values of `TotalPrice` in `Orders` by more refined ranges based on the first digit and the number of digits. Assuming that values are always non-negative (and most numeric values in databases are non-negative), the expression for the upper and lower bound is:

```
SELECT lowerbound, upperbound, COUNT(*) as numorders, MIN(val), MAX(val)
FROM (SELECT (FLOOR(val / POWER(10.0, SIGN(numdigits)*(numdigits - 1))) *
              POWER(10.0, SIGN(numdigits)*(numdigits - 1))
             ) as lowerbound,
            (FLOOR(1 + (val / POWER(10.0, SIGN(numdigits)*(numdigits - 1)))) *
              POWER(10.0, SIGN(numdigits)*(numdigits - 1))
             ) as upperbound, o.*
      FROM (SELECT (LEN(CAST(FLOOR(ABS(TotalPrice)) as INT)) *
                    SIGN(FLOOR(TotalPrice))) as numdigits,
                   TotalPrice as val
            FROM Orders o
           ) o
     ) o
GROUP BY lowerbound, upperbound
ORDER BY lowerbound
```

This query uses two subqueries. The innermost calculates `numdigits` and the middle calculates `lowerbound` and `upperbound`. In the complicated expressions for the bounds, the `SIGN()` function is used to handle the case when the number of digits is zero.

Breaking Numeric Values into Equal-Sized Groups

Equal-sized ranges are perhaps the most useful type of ranges. For instance, the middle value in a list (the median) splits the list into two equal-sized groups.

Table 2-4: Ranges of Values for TotalPrice in Orders Table by First Digit and Number of Digits

LOWER BOUND	UPPER BOUND	NUMBER OF ORDERS	MINIMUM TOTALPRICE	MAXIMUM TOTALPRICE
$0	$1	9,130	$0.00	$0.64
$1	$2	4	$1.75	$1.95
$2	$3	344	$2.00	$2.95
$3	$4	2	$3.50	$3.75
$4	$5	13	$4.00	$4.95
$5	$6	152	$5.00	$5.97
$6	$7	1,591	$6.00	$6.99
$7	$8	2,015	$7.00	$7.99
$8	$9	1,002	$8.00	$8.99
$9	$10	1,595	$9.00	$9.99
$10	$20	54,382	$10.00	$19.99
$20	$30	46,434	$20.00	$29.99
$30	$40	20,997	$30.00	$39.99
$40	$50	9,378	$40.00	$49.98
$50	$60	6,366	$50.00	$59.99
$60	$70	3,629	$60.00	$69.99
$70	$80	2,017	$70.00	$79.99
$80	$90	3,257	$80.00	$89.99
$90	$100	1,661	$90.00	$99.99
$100	$200	16,590	$100.00	$199.98
$200	$300	1,272	$200.00	$299.97
$300	$400	6,083	$300.00	$399.95
$400	$500	1,327	$400.00	$499.50
$500	$600	1,012	$500.00	$599.95
$600	$700	670	$600.00	$697.66
$700	$800	393	$700.00	$799.90
$800	$900	320	$800.00	$895.00
$900	$1,000	361	$900.00	$999.00
$1,000	$2,000	731	$1,000.00	$1,994.00
$2,000	$3,000	155	$2,000.00	$2,995.00
$3,000	$4,000	54	$3,000.00	$3,960.00
$4,000	$5,000	20	$4,009.50	$4,950.00
$5,000	$6,000	10	$5,044.44	$5,960.00
$6,000	$7,000	12	$6,060.00	$6,920.32
$8,000	$9,000	1	$8,830.00	$8,830.00
$9,000	$10,000	3	$9,137.09	$9,848.96

Which value is in the middle? Unfortunately, there is no aggregation function for calculating the median, as there is for the average.

One approach is to use `ROW_NUMBER()`. If there are nine rows of data and with ranks one through nine, the median value is the value on the fifth row.

Finding quintiles and deciles is the same process as finding the median. Quintiles break numeric ranges into five equal-sized groups; four breakpoints are needed to do this—the first for the first 20% of the rows; the second for the next 20%, and so on. Creating deciles is the same process but with nine breakpoints instead.

The following query provides the framework for finding quintiles, using the ranking window function `ROW_NUMBER()`:

```
SELECT MAX(CASE WHEN seqnum <= cnt * 0.2 THEN <val> END) as break1,
       MAX(CASE WHEN seqnum <= cnt * 0.4 THEN <val> END) as break2,
       MAX(CASE WHEN seqnum <= cnt * 0.6 THEN <val> END) as break3,
       MAX(CASE WHEN seqnum <= cnt * 0.8 THEN <val> END) as break4
FROM (SELECT ROW_NUMBER() OVER (ORDER BY <val>) as seqnum,
             COUNT(*) OVER () as cnt,
             <val>
      FROM <table>) t
```

It works by enumerating the rows in order by the desired column, and comparing the resulting row number with the total number of rows. This technique works for any type of column. For instance, it can find the values used for breaking up date ranges and character strings into equal-sized groups.

More Values to Explore—Min, Max, and Mode

Columns have other interesting characteristics. This section discusses extreme values and the most common value.

Minimum and Maximum Values

SQL makes it quite easy to find the minimum and maximum values in a table for any data type. The minimum and maximum values for strings are based on the alphabetic ordering of the values. The query is simply:

```
SELECT MIN(<col>), MAX(<col>)
FROM <tab>
```

A related question is the frequency of maximum and minimum values in a particular column. Answering this question uses a subquery in the `SELECT` clause of the query:

```
SELECT SUM(CASE WHEN <col> = minv THEN 1 ELSE 0 END) as freqminval,
       SUM(CASE WHEN <col> = maxv THEN 1 ELSE 0 END) as freqmaxval
FROM <tab> t CROSS JOIN
     (SELECT MIN(<col>) as minv, MAX(<col>) as maxv
      FROM <tab>) vals
```

This query uses the previous query as a subquery to calculate the minimum and maximum values. Because there is only one row, the `CROSS JOIN` operator is used for the join. This technique can be extended. For instance, it might be interesting to count the number of values within 10% of the maximum or minimum value for a numeric value. This calculation is as simple as multiplying `MAX(<col>)` by 0.9 and `MIN(<col>)` by 1.1 and replacing the "=" with ">=" and "<=" respectively.

Sometimes, the entire row containing the maximum or minimum value is of interest. For this purpose, use `ORDER BY`. For instance, the following query gets a row that has the maximum value for a given column:

```
SELECT TOP 1 t.*
FROM <tab> t
ORDER BY col DESC
```

For the minimum value, change the last line to `ORDER BY col`.

The Most Common Value (Mode)

The most common value is called the *mode*. The mode differs from other measures that we've looked at so far. There is only one maximum, minimum, and average and generally only one median. However, there can be many modes. A common, but not particularly interesting, example is the primary key of a table, which is never repeated. All values have a frequency of one, so all values are modes.

Calculating the mode in standard SQL is a bit cumbersome. The next sections show two different approaches to the calculation.

Calculating Mode Using Basic SQL

Calculating the mode starts with calculating the frequency of values in a column:

```
SELECT <col>, COUNT(*) as freq
FROM <tab>
GROUP BY <col>
ORDER BY freq
```

The mode is the last row (or the first row if the list is sorted in descending order). To get this row, you can use `SELECT TOP 1` instead of just `SELECT`.

What column values have the same frequency as the maximum column frequency? A subquery can help answer this question:

```
SELECT <col>, COUNT(*) as freq
FROM <tab>
GROUP BY <col>
HAVING COUNT(*) = (SELECT TOP 1 COUNT(*) as freq
                   FROM <tab>
                   GROUP BY <col>
                   ORDER BY COUNT(*) DESC)
```

In this query, the `HAVING` clause does almost all the work. It selects the groups (column values) whose frequency is the same as the largest frequency. What is the largest frequency? That is calculated by the subquery. The result is a list of the values whose frequencies match the maximum frequency, a list of the modes.

If, instead, we were interested in the values with the smallest frequency, the `MAX(freq)` expression would be changed to `MIN(freq)`. Such values could be considered the antimode values.

This query accomplishes the task at hand. However, it is rather complex, with multiple levels of subqueries and two references to the table. It is easy to make mistakes when writing such queries, and complex queries are harder to optimize for performance. The next section offers a simpler alternative.

Calculating Mode Using Window Functions

The following query uses `MAX()` as a window function to find the mode:

```
SELECT t.*
FROM (SELECT <col>, COUNT(*) as freq, MAX(COUNT(*)) OVER () as maxfreq
      FROM <tab>
      GROUP BY <col>
     ) t
WHERE freq = maxfreq
```

Note that the `COUNT(*)` is the argument to the window function `MAX() OVER ()`. This expression calculates the maximum of the count, which is the maximum frequency. The outermost `WHERE` selects the rows where the frequency matches the maximum.

Exploring String Values

String values pose particular challenges for data exploration because they can take on almost any value. This is particularly true for free-form strings, such as addresses and names, which may not be cleaned. This section looks at exploring the length and characters in strings.

Histogram of Length

A simple way to get familiar with string values is to do a histogram of the length of the values. *What is the length of values in the* `City` *column in the* `Orders` *table?*

```
SELECT LEN(City) as length, COUNT(*) as numorders, MIN(City), MAX(City)
FROM Orders o
GROUP BY LEN(City)
ORDER BY length
```

This query provides not only a histogram of the lengths, but also examples of two values—the minimum and maximum values for each length. For the `City` column, there are lengths from 0 to 20, which is the maximum length the column stores.

Strings Starting or Ending with Spaces

Spaces at the beginning of string values can cause unexpected problems. The value " NY" is not the same as "NY," so a comparison operation or join might fail—even though the values look the same to humans. Spaces at the end of strings pose less of a problem because they are typically ignored for equality comparisons.

The following query answers the question: *How many times do the values in the column have spaces at the beginning or end of the value?*

```
SELECT COUNT(*) as numorders
FROM Orders o
WHERE City IS NOT NULL AND LEN(City) <> LEN(LTRIM(RTRIM(City)))
```

This query works by stripping spaces from the beginning and end of the column, and then comparing the lengths of the stripped and unstripped values.

Handling Upper- and Lowercase

Databases can be either case sensitive or case insensitive. Case sensitive means that upper- and lowercase characters are considered different; case insensitive means they are the same. Don't be confused by case sensitivity in strings versus case sensitivity in syntax. SQL keywords can be in any case ("`SELECT`," "`select`," "`Select`"). This discussion only refers to how values in columns are treated.

For instance, in a case-insensitive database, the following values would all be equal to each other:

- FRED
- Fred
- fRed

By default, most databases are case insensitive. However, this can be changed by setting a global option or by passing hints to a particular query (such as using the `COLLATE` keyword in SQL Server).

In a case-sensitive database, the following query answers the question: *How often are the values all uppercase, all lowercase, or mixed case?*

```
SELECT SUM(CASE WHEN City = UPPER(City) THEN 1 ELSE 0 END) as uppers,
       SUM(CASE WHEN City = LOWER(City) THEN 1 ELSE 0 END) as lowers,
       SUM(CASE WHEN City NOT IN (LOWER(City), UPPER(City))
                THEN 1 ELSE 0 END) as mixed
FROM Orders o
```

In a case-insensitive database, the first two values are the same and the third is zero. In a case-sensitive database, the three add up to the total number of rows.

What Characters Are in a String?

Sometimes, it is interesting to know exactly what characters are in strings. For instance, do email addresses provided by customers contain characters that they should not? Such a question naturally leads to which characters are actually in the values.

SQL is not designed to answer this question, at least in a simple way. Fortunately, it is possible to make an attempt. The answer starts with a simpler question: *What characters are in the first position of the string?*

```
SELECT LEFT(City, 1) as onechar, ASCII(LEFT(City, 1)) as asciival,
       COUNT(*) as numorders
FROM Orders o
GROUP BY LEFT(City, 1)
ORDER BY onechar
```

The returned data has three columns: the character, the number that represents the character (called the ASCII value), and the number of times that the character occurs as the first character in the `City` column. The ASCII value is useful for distinguishing among characters that might look the same, such as a space and a tab.

WARNING **When looking at individual characters, unprintable characters and space characters (space and tabs) look the same. To see what character is really there, use the `ASCII()` function.**

The following query extends this example to look at the first two characters in the `City` column:

```
SELECT onechar, ASCII(onechar) as asciival, COUNT(*) as cnt
FROM ((SELECT SUBSTRING(City, 1, 1) as onechar
       FROM Orders WHERE LEN(City) >= 1)
```

CHARACTERS AND COLLATIONS

Character strings in SQL are more complex than they appear. This is because SQL strives to support all sorts of writing systems. The Latin characters used in English are a simple example. Most European languages use a similar system, although augmented by various accented characters and the occasional special character. And then there are hundreds of alphabets around the world that are also supported. Some read from right to left, some from left to right. And a language such as Chinese doesn't technically have an alphabet; instead it has tens of thousands of characters.

Characters themselves are represented as combinations of zeros and ones inside the computer. Three interrelated concepts are useful for understanding this representation. The first is the *character set* or *character map* which refers to what the bits mean. A very common system for English letters is ASCII, where, for instance, the bits that represent the number 65 represent the letter "A."

The *collation* refers to how the characters are ordered and whether or not two particular characters are equal. For instance, capital "A" and lowercase "a" may be equal, when the collation is case insensitive. The third concept is the *font*, which refers to how the character set is rendered on the screen or printed. SQL doesn't know about fonts although Excel does.

SQL has four types for storing character strings of a given length:

- `CHAR()`
- `VARCHAR()`
- `NCHAR()`
- `NVARCHAR()`

The first of these is a fixed length string. Shorter values stored in a `CHAR()` field are padded at the end with spaces. So, "NY" would be "NY___" if stored in a `CHAR(5)` field. Normally, fixed-length characters are used for short codes, especially when all codes are the same length. `VARCHAR()` can store variable length strings. These are not padded on the right with spaces. Characters in these fields are stored using a single byte per character.

The last two are types for national characters; these types require more space to store a given value than `CHAR()` or `VARCHAR()`. However, they are much more flexible and can store characters from a mix of alphabets or from complicated writing systems such as Chinese and Japanese.

Within the database, the "collation" of the column determines both the collation (rules for comparison) and the character set (rules for presentation). Characters sets are typically customized for languages (so accented characters are represented) and for the natural ordering within the language.

Collations and character sets affect queries, from comparisons to ordering and aggregation. Fortunately, the default collations are usually quite sufficient. They become particularly useful (and annoying) when using databases for multilingual applications. Happily the database supports them. For most purposes, the only interest in collations is determining which is needed for case-sensitive or case-insensitive comparisons.

```
      UNION ALL
      (SELECT SUBSTRING(City, 2, 1) as onechar
       FROM Orders WHERE LEN(City) >= 2)
     ) cl
GROUP BY onechar
ORDER BY onechar
```

This query combines all the first characters and all the second characters together, using `UNION ALL` in the subquery. It then groups this collection of characters together, returning the final result. Extending this query to all 20 characters in the city is a simple matter of adding more subqueries to the `UNION ALL`.

A variation of this query might be more efficient under some circumstances. This variation pre-aggregates each of the subqueries. Rather than just putting all the characters together and then aggregating, it calculates the frequencies for the first position and then the second position, and then combines the results:

```
SELECT onechar, ASCII(onechar) as asciival, SUM(cnt) as cnt
FROM ((SELECT SUBSTRING(City, 1, 1) as onechar, COUNT(*) as cnt
       FROM Orders WHERE LEN(City) >= 1
       GROUP BY SUBSTRING(City, 1, 1) )
      UNION ALL
      (SELECT SUBSTRING(City, 2, 1) as onechar, COUNT(*) as cnt
       FROM Orders WHERE LEN(City) >= 2
       GROUP BY SUBSTRING(City, 2, 1) )
     ) cl
GROUP BY onechar
ORDER BY onechar
```

The choice between the two forms is a matter of convenience and efficiency, both in writing the query and in running it.

What if the original question were: *How often does a character occur in the first position versus the second position of a string?* This is quite similar to the original question, and the answer is to use conditional aggregation based on the position of the character:

```
SELECT onechar, ASCII(onechar) as asciival, COUNT(*) as cnt,
       SUM(CASE WHEN pos = 1 THEN 1 ELSE 0 END) as pos_1,
       SUM(CASE WHEN pos = 2 THEN 1 ELSE 0 END) as pos_2
FROM ((SELECT SUBSTRING(City, 1, 1) as onechar, 1 as pos
       FROM Orders o WHERE LEN(City) >= 1 )
      UNION ALL
      (SELECT SUBSTRING(City, 2, 1) as onechar, 2 as pos
       FROM Orders o WHERE LEN(City) >= 2)
     ) a
GROUP BY onechar
ORDER BY onechar
```

This variation also works using the pre-aggregated subqueries.

Exploring Values in Two Columns

Comparing values in more than one column is an important part of data exploration and data analysis. This section focuses on description. Do two states differ by sales? Do customers who purchase more often have larger average purchases? Whether the comparison is statistically significant is covered in the next chapter.

What Are Average Sales by State?

The following two questions are good examples of comparing a numeric value within a categorical value:

- What is the average order total price by state?
- What is the average zip code population in a state?

SQL is particularly adept at answering such questions using aggregations.

The following query provides the average sales by state:

```
SELECT State, AVG(TotalPrice) as avgtotalprice
FROM Orders
GROUP BY State
ORDER BY avgtotalprice DESC
```

This example uses the aggregation function `AVG()` to calculate the average.

The following expression could also have been used:

```
SELECT state, SUM(TotalPrice)/COUNT(*) as avgtotalprice
```

Although the two methods seem to do the same thing, there is a subtle difference between them, because they handle `NULL` values differently. In the first example, `NULL` values are ignored. In the second, `NULL` values contribute to the `COUNT(*)`, but not to the `SUM()`. Replacing `COUNT(*)` with `COUNT(TotalPrice)` fixes this, by counting the number of values that are not `NULL`.

Even with the fix, there is still a subtle difference when all the values are `NULL`. The `AVG()` returns a `NULL` value in this case. The explicit division returns a divide-by-zero error. To fix this, replace the `0` with `NULL`: `NULLIF(COUNT(TotalPrice), 0)`.

TIP **Two ways of calculating an average look similar and often return the same result. However,** `AVG(<col>)` **and** `SUM(<col>)/COUNT(*)` **treat** `NULL` **values differently.**

How Often Are Products Repeated within a Single Order?

A reasonable assumption is that each product has only one order line in an order, regardless of the number of units ordered; the multiple instances are represented

by the column NumUnits rather than by separate rows in OrderLines. There are several different methods to validate this assumption.

Direct Counting Approach

The first approach directly answers the question: *How many different order lines within an order contain the same product?* This is a simple counting query, using two different columns instead of one:

```
SELECT cnt, COUNT(*) as numorders, MIN(OrderId), MAX(OrderId)
FROM (SELECT OrderId, ProductId, COUNT(*) as cnt
      FROM OrderLines ol
      GROUP BY OrderId, ProductId
     ) op
GROUP BY cnt
ORDER BY cnt
```

Here, cnt is the number of times that a given OrderId and ProductId appear together in a row in OrderLines.

The results show that some products are repeated within the same order, up to a maximum of 40 times. This leads to more questions. What are some examples of orders where duplicate products occur? For this, the minimum and maximum OrderId provide examples.

Which products are more likely to occur multiple times within an order? A result table with the following information would help in answering this question:

- ProductId, to identify the product
- Number of orders containing the product any number of times
- Number of orders containing the product more than once

These second and third columns compare the occurrence of the given product overall with the multiple occurrence of the product within an order.

The following query does the calculation:

```
SELECT ProductId, COUNT(*) as numorders,
       SUM(CASE WHEN cnt > 1 THEN 1 ELSE 0 END) as nummultiorders
FROM (SELECT OrderId, ProductId, COUNT(*) as cnt
      FROM OrderLines ol
      GROUP BY OrderId, ProductId
     ) op
GROUP BY ProductId
ORDER BY numorders DESC
```

The results (which have thousands of rows) indicate that some products are, indeed, more likely to occur multiple times within an order than other products.

However, many products occur multiple times in a single order, so the duplication is not caused by just a handful of errant products.

Comparison of Distinct Counts to Overall Counts

Another approach to answering the question *"How often are products repeated in an order?"* is to consider the number of order lines in an order compared to the number of different products in the same order. That is, calculate the number of order lines and the number of distinct product IDs in each order; these numbers are the same when an order has no duplicate products.

One way of doing the calculation is using `COUNT(DISTINCT)`:

```
SELECT OrderId, COUNT(*) as numlines,
       COUNT(DISTINCT ProductId) as numproducts
FROM OrderLines ol
GROUP BY OrderId
HAVING COUNT(*) > COUNT(DISTINCT ProductId)
```

The `HAVING` clause chooses only orders that have at least one product on multiple order lines.

Another approach uses a subquery:

```
SELECT OrderId, SUM(numproductlines) as numlines,
       COUNT(*) as numproducts
FROM (SELECT OrderId, ProductId, COUNT(*) as numproductlines
      FROM OrderLines ol
      GROUP BY OrderId, ProductId) op
GROUP BY OrderId
HAVING SUM(numproductlines) > COUNT(*)
```

The subquery aggregates the order lines by `OrderId` and `ProductId`. This intermediate result can be used to count both the number of products and the number of order lines. In general, a query using `COUNT(DISTINCT)`can also be rewritten to use a subquery, but `COUNT(DISTINCT)` is more convenient.

There are 4,878 orders that have more order lines than products, indicating that at least one product occurs on multiple lines in the order. However, the results from the query do not give an idea of what might be causing this.

Perhaps orders with a lot of products are the culprit. The following query calculates the number of orders that have more than one product broken out by the number of lines in the order:

```
SELECT numlines, COUNT(*) as numorders,
       SUM(CASE WHEN numproducts < numlines THEN 1 ELSE 0
           END) as nummultiorders,
       AVG(CASE WHEN numproducts < numlines THEN 1.0 ELSE 0
           END) as ratiomultiorders,
       MIN(OrderId), MAX(OrderId)
```

```
FROM (SELECT OrderId, COUNT(DISTINCT ProductId) as numproducts,
             COUNT(*) as numlines
      FROM OrderLines ol
      GROUP BY OrderId
     ) op
GROUP BY numlines
ORDER BY numorders;
```

This query uses `COUNT(DISTINCT)` along with `COUNT()` to calculate the number of products and order lines within a query.

Table 2-5 shows the first few rows of the results. The proportion of multiorders increases as the size of the order increases. However, for all order sizes, many orders still have nonrepeating products. Based on this information, it seems that having multiple lines for a single product is a function of having larger orders, rather than being related to the particular products in the order.

Which State Has the Most American Express Users?

Overall, about 24.6% of the orders are paid by American Express (payment type AE). *Does this proportion vary much by state?* The following query answers this question:

```
SELECT State, COUNT(*) as numorders,
       SUM(CASE WHEN PaymentType = 'AE' THEN 1 ELSE 0 END) as numae,
       AVG(CASE WHEN PaymentType = 'AE' THEN 1.0 ELSE 0 END) as avgae
FROM Orders o
GROUP BY State
HAVING COUNT(*) >= 100
ORDER BY avgae DESC
```

Table 2-5: Number of Products Per Order by Number of Lines in Order (First Ten Rows)

		ORDERS WITH MORE LINES THAN PRODUCTS	
LINES IN ORDER	NUMBER OF ORDERS	NUMBER	%
1	139,561	0	0.0%
2	32,758	977	3.0%
3	12,794	1,407	11.0%
4	3,888	894	23.0%
5	1,735	532	30.7%
6	963	395	41.0%
7	477	223	46.8%
8	266	124	46.6%
9	175	93	53.1%
10	110	65	59.1%

Table 2-6: Percent of American Express Payment for Top Ten States with Greater Than 100 Orders

STATE	# ORDERS	# AE	% AE
GA	2,865	1,141	39.8%
PR	168	61	36.3%
LA	733	233	31.8%
FL	10,185	3,178	31.2%
NY	53,537	16,331	30.5%
DC	1,969	586	29.8%
NJ	21,274	6,321	29.7%
MS	215	63	29.3%
MT	111	29	26.1%
UT	361	94	26.0%

This query calculates the number and percentage of orders paid by American Express by customers in each state, and then returns the states ordered with the highest proportion at the top. The query only chooses states that have at least 100 orders, in order to eliminate specious state codes. Notice the use of 1.0 instead of 1 for the average. Some databases (notably SQL Server) do integer arithmetic on integers. So, the average of 1 and 2 is 1 rather than 1.5. Table 2-6 shows the top ten states by this proportion.

From Summarizing One Column to Summarizing All Columns

So far, the exploratory data analysis has focused on summarizing values in a single column. This section first combines the various results into a single summary for a column. It then extends this summary from a single column to all columns in a table. In the process, we use SQL (or alternatively Excel) to generate a SQL query, which generates the summaries.

Good Summary for One Column

For exploring data, the following information is a good summary for a single column:

- The number of distinct values in the column
- Minimum and maximum values
- An example of the most common value (the *mode*)
- An example of the least common value (the *antimode*)

- Frequency of the minimum and maximum values
- Frequency of the mode and antimode
- Number of values that occur only one time
- Number of modes (because the most common value is not necessarily unique)
- Number of antimodes

These summary statistics are defined for all data types. Additional information might be of interest for other data types, such as the minimum and maximum length of strings, the average value of a numeric, and the number of times when a date has no time component.

The following query calculates these values for `State` in `Orders`:

```
WITH osum as (
      SELECT 'state' as col, State as val, COUNT(*) as freq
      FROM Orders o
      GROUP BY State
     )
SELECT osum.col, COUNT(*) as numvalues,
       MAX(freqnull) as freqnull,
       MIN(minval) as minval,
       SUM(CASE WHEN val = minval THEN freq ELSE 0 END) as numminvals,
       MAX(maxval) as maxval,
       SUM(CASE WHEN val = maxval THEN freq ELSE 0 END) as nummaxvals,
       MIN(CASE WHEN freq = maxfreq THEN val END) as mode,
       SUM(CASE WHEN freq = maxfreq THEN 1 ELSE 0 END) as nummodes,
       MAX(maxfreq) as modefreq,
       MIN(CASE WHEN freq = minfreq THEN val END) as antimode,
       SUM(CASE WHEN freq = minfreq THEN 1 ELSE 0 END) as numantimodes,
       MAX(minfreq) as antimodefreq,
       SUM(CASE WHEN freq = 1 THEN freq ELSE 0 END) as numuniques
FROM osum CROSS JOIN
     (SELECT MIN(freq) as minfreq, MAX(freq) as maxfreq,
             MIN(val) as minval, MAX(val) as maxval,
             SUM(CASE WHEN val IS NULL THEN freq ELSE 0 END) as freqnull
      FROM osum
     ) summary
GROUP BY osum.col
```

This query follows a simple logic. The CTE `osum` summarizes the data by `State`. The second subquery summarizes the summary, producing values for:

- Minimum and maximum frequency
- Minimum and maximum values
- Number of `NULL` values

The outer query combines these two, making judicious use of the `CASE` statement.

The results for `State` are as follows:

- Number of values: 92
- Minimum value: ""
- Maximum value: YU
- Mode: NY
- Antimode: BD
- Frequency of Nulls: 0
- Frequency of Min: 1,119
- Frequency of Max: 2
- Frequency of Mode: 53,537
- Frequency of Antimode: 1
- Number of Unique Values: 14
- Number of Modes: 1
- Number of Antimodes: 14

As mentioned earlier, this summary works for all data types.

The query is set up so only the first row in the CTE needs to change for another column. So, it is easy to get results for, say, `TotalPrice`:

- Number of values: 7,653
- Minimum value: $0.00
- Maximum value: $9,848.96
- Mode: $0.00
- Antimode: $0.20
- Frequency of Nulls: 0
- Frequency of Min: 9,128
- Frequency of Max: 1
- Frequency of Mode: 9,128
- Frequency of Antimode: 1
- Number of Unique Values: 4,115
- Number of Modes: 1
- Number of Antimodes: 4,115

The most common value of `TotalPrice` is $0. One reason for this is that all other values have both dollars and cents in their values. The proportion of orders with

$0 value is small. This suggests doing the same analysis but using only the dollar amount of `TotalPrice`. This is accomplished by replacing the `TotalPrice as val` with `FLOOR(TotalPrice) as val`.

The next two sections approach the question of how to generate this information for all columns in a table. The strategy is to query the database for all columns in a table and then use SQL or Excel to write the query.

Query to Get All Columns in a Table

Most databases store information about their columns and tables in special system tables and views. The following query returns the table name and column names of all the columns in the `Orders` table, using a common syntax:

```
SELECT (table_schema + '.' + table_name) as table_name, column_name,
       ordinal_position
FROM INFORMATION_SCHEMA.COLUMNS c
WHERE LOWER(table_name) = 'orders'
```

See the Appendix for mechanisms in other databases.

The results are in Table 2-7, which is simply the table name and list of columns in the table. The view `INFORMATION_SCHEMA.COLUMNS` also contains information that the query does not use, such as whether the column allows `NULL` values and the type of the column.

Table 2-7: Column Names in Orders

TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION
Orders	ORDERID	1
Orders	CUSTOMERID	2
Orders	CAMPAIGNID	3
Orders	ORDERDATE	4
Orders	CITY	5
Orders	STATE	6
Orders	ZIPCODE	7
Orders	PAYMENTTYPE	8
Orders	TOTALPRICE	9
Orders	NUMORDERLINES	10
Orders	NUMUNITS	11

Using SQL to Generate Summary Code

The goal is to summarize all the columns in a table, using an information summary subquery for each column. Such a query has the following pattern for `Orders`:

```
(INFORMATION SUBQUERY for orderid)
UNION ALL (INFORMATION SUBQUERY for customerid)
UNION ALL (INFORMATION SUBQUERY for campaignid)
UNION ALL (INFORMATION SUBQUERY for orderdate)
UNION ALL (INFORMATION SUBQUERY for city)
UNION ALL (INFORMATION SUBQUERY for state)
UNION ALL (INFORMATION SUBQUERY for zipcode)
UNION ALL (INFORMATION SUBQUERY for paymenttype)
UNION ALL (INFORMATION SUBQUERY for totalprice)
UNION ALL (INFORMATION SUBQUERY for numorderlines)
UNION ALL (INFORMATION SUBQUERY for numunits)
```

The information subquery is similar to the earlier version, with the mode and antimode values removed (just to simplify the query for explanation).

There are four other modifications to the query. The first is to remove the CTE. A `UNION ALL` query can have only one `WITH` clause, instead of one for each subquery. The second is to include a placeholder called `<start>` at the beginning. The third is to convert the minimum and maximum values to strings because all values in a given column need to be of the same type for the `UNION ALL`. The resulting query has the general form:

```
<start> SELECT '<col>' as colname, COUNT(*) as numvalues,
       MAX(freqnull) as freqnull,
       CAST(MIN(minval) as VARCHAR(255)) as minval,
       SUM(CASE WHEN <col> = minval THEN freq ELSE 0 END) as numminvals,
       CAST(MAX(maxval) as VARCHAR(255)) as maxval,
       SUM(CASE WHEN <col> = maxval THEN freq ELSE 0 END) as nummaxvals,
       SUM(CASE WHEN freq = 1 THEN freq ELSE 0 END) as numuniques
FROM (SELECT <col>, COUNT(*) as freq
      FROM <tab>
      GROUP BY <col>) osum CROSS JOIN
     (SELECT MIN(<col>) as minval, MAX(<col>) as maxval,
             SUM(CASE WHEN <col> IS NULL THEN 1 ELSE 0 END) as freqnull
      FROM <tab>
     ) summary
```

The next step is to put this query, as long as it is, into a single line.

To construct the final query, we'll use the string function `REPLACE()` to put in the column and table names:

```
SELECT REPLACE(REPLACE(REPLACE('<start> SELECT ''<col>'' as colname,
COUNT(*) as numvalues, MAX(freqnull) as freqnull, CAST(MIN(minval) as
VARCHAR(255)) as minval, SUM(CASE WHEN <col> = minval THEN freq ELSE 0
```

```
END) as numminvals, CAST(MAX(maxval) as VARCHAR(255)) as maxval, SUM
(CASE WHEN <col> = maxval THEN freq ELSE 0 END) as nummaxvals, SUM(CASE
WHEN freq = 1 THEN 1 ELSE 0 END) as numuniques FROM (SELECT <col>,
COUNT(*) as freq FROM <tab> GROUP BY <col>) osum CROSS JOIN (SELECT
MIN(<col>) as minval, MAX(<col>) as maxval, SUM(CASE WHEN <col> IS NULL
THEN 1 ELSE 0 END) as freqnull FROM <tab>) summary',
                              '<col>', column_name),
                        '<tab>', table_name),
               '<start>',
                 (CASE WHEN ordinal_position = 1 THEN ''
                       ELSE 'UNION ALL' END))
FROM (SELECT table_name, column_name, ordinal_position
      FROM INFORMATION_SCHEMA.COLUMNS
      WHERE lower(table_name) = 'orders') tc
```

This query replaces three placeholders in the query string with appropriate values. The "`<col>`" string gets replaced with the column name, which comes from `INFORMATION_SCHEMA.COLUMNS`. The "`<tab>`" string gets replaced with the table name. And, the "`<starting>`" string gets "`UNION ALL`" for all but the first row. That is how the different subqueries are combined.

This query can be pasted into the query tool. Table 2-8 shows the results from running the resulting query.

Note that we could also construct this query in Excel. This starts by querying the metadata table for the names of the columns in the table. `SUBSTITUTE()` can then be used to do the replacements and get the final query.

Table 2-8: Information about the Columns in Orders Table

COLNAME	# VALUES	# NULL	# MINIMUM VALUE	# MAXIMUM VALUE	# UNIQUE
OrderId	192,983	0	1	1	192,983
CustomerId	189,560	0	3,424	1	189,559
CampaignId	239	0	5	4	24
OrderDate	2,541	0	181	2	0
City	12,825	0	17	5	6,318
State	92	0	1,119	2	14
ZipCode	15,579	0	144	1	5,954
PaymentType	6	0	313	77,017	0
TotalPrice	7,653	0	9,128	1	4,115
NumOrderLines	41	0	139,561	1	14
NumUnits	142	0	127,914	1	55

Lessons Learned

Databases are well suited to data exploration because databases are close to the data. Most relational databases are inherently parallel—meaning they can take advantage of multiple processors and multiple disks—so a database is often the best choice in terms of performance as well. Excel charting is a useful companion because it is familiar to business people and charts are a powerful way to communicate results. This chapter introduces several types of charts including column charts, line charts, scatter plots, and sparklines.

Data exploration starts by investigating the values stored in various columns in tables. Histograms are a good way to see distributions of values in particular columns, although numeric values often need to be grouped to see their distributions. There are various ways of grouping numeric values into ranges, including *tiling*—creating equal-sized groups such as quintiles and deciles.

Various other metrics are of interest in describing data in columns. The most common value is called the *mode*, which can be calculated in SQL.

Ultimately, it is more efficient to investigate all columns at once rather than each column one at a time. The chapter ends with a mechanism for creating a single query to summarize all columns at the same time. This method uses SQL or Excel to create a complex query, which is then run to get summaries for all the columns in a table.

The next chapter moves from just exploring and looking at the data to determining whether patterns in the data are, in fact, statistically significant.