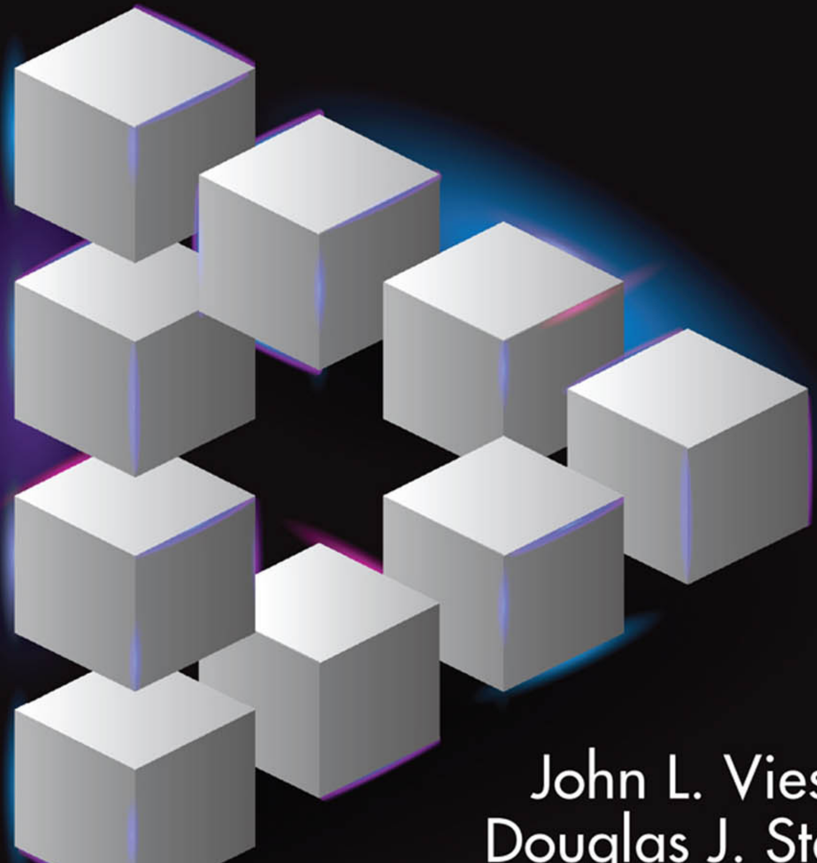
Effective SOFTWARE DEVELOPMENT SERIES
Scott Meyers, Consulting Editor

Effective SQL

61 Specific Ways to Write Better SQL



John L. Viescas
Douglas J. Steele
Ben G. Clothier

Foreword by Keith W. Hare

Praise for *Effective SQL*

"Given the reputation of the authors, I expected to be impressed. Impressed doesn't cover it, though. I was blown away! Most SQL books tell you 'how.' This one tells you 'why.' Most SQL books separate database design from implementation. This one integrates design considerations into every facet of SQL use. Most SQL books sit on my shelf. This one will live on my desk."

—Roger Carlson, Microsoft Access MVP (2006–2015)

"It can be easy to learn the basics of SQL, but it is very difficult to build accurate and efficient SQL, especially for critical systems with complex requirements. But now, with this great new book, you can get up to speed and write effective SQL much more quickly, no matter which DBMS you use."

—Craig S. Mullins, Mullins Consulting, Inc., DB2 Gold Consultant and IBM Champion for Analytics

"This is a great book. It is written in language that can be understood by a relative beginner and yet contains tips and tricks that will benefit the most hardened workhorse. It will therefore appeal to readers across the whole range of expertise and should be in the library of anybody who is seriously concerned with designing, managing, or programming databases."

—Graham Mandeno, database consultant and Microsoft MVP (1996–2015)

"This book is an excellent resource for database designers and developers working with relational and SQL-based databases—it's an easy read with great examples that combine theory with practical examples seamlessly. Examples for top relational databases Oracle, DB2, SQL Server, MySQL, and PostgreSQL are included throughout. The book walks the reader through sophisticated techniques to deal with things such as hierarchical data and tally tables, along with explanations of the inner workings and performance implications of SQL using `GROUP BY`, `EXISTS`, `IN`, correlated and non-correlated subqueries, window functions, and joins. The tips you won't find anywhere else, and the fun examples help to make this book stand out from the crowd."

—Tim Quinlan, database architect and Oracle Certified DBA

"This book is good for those who need to support multiple dialects of SQL. It's divided up into stand-alone items that you just grab and go. I have been doing SQL in various flavors since 1992 and even I picked up a few things."

—Tom Moreau, Ph.D., SQL Server MVP (2001–2012)

"This book is a powerful, compact, and easily understandable presentation of how to use SQL—it shows the application of SQL to real-world questions in order to teach the construction of queries, and it explains the relationship of 'how data is stored' to 'how data is queried' so that you obtain results successfully and effectively."

—Kenneth D. Snell, Ph.D., database consultant and former Microsoft Access MVP

excluded those customers. This can potentially result in far more I/Os than you anticipated. You can learn more about those considerations in Item 46, "Understand how the execution plan works." Although this is a somewhat oversimplified example, it is fairly easy to create a view that the optimizer simply cannot inline when it is referenced in other views. Worse, there would be more than one way to create such views that would prevent inlining. Finally, the optimizer generally does a better job when it is given a simpler query expression that asks for exactly the data it actually needs.

For those reasons, it is best to avoid creating views on views. If you need a different presentation of the view, create a new view that directly references the base tables with the appropriate filters or groupings applied. You can also embed subqueries in a view, which can be useful in making the aggregated calculations "private" to the view. This approach helps to prevent proliferation of several views that are not directly usable, making the database solution much more maintainable. Refer to Item 42, "If possible, use common table expressions instead of subqueries," for additional techniques.

Things to Remember

- ✦ Use views to structure data in a way that users will find natural or intuitive.
- ✦ Use views to restrict access to the data such that users can see (and sometimes modify) exactly what they need and no more. Remember to use `WITH CHECK OPTION` when necessary.
- ✦ Use views to hide and reuse complex queries.
- ✦ Use views to summarize data from various tables that can be used to generate reports.
- ✦ Use views to implement and enforce naming and coding standards, especially when working with legacy database designs that need to be updated.

Item 19: Use ETL to Turn Nonrelational Data into Information

Extract, Transform, Load (ETL) is a set of procedures or tools you can use to **E**xtract data from an external source, **T**ransform it to conform to relational design rules or to conform to other requirements, then **L**oad it into your database for further use or analysis. Nearly all database systems provide various utilities to aid in this process. These utilities are, quite simply, a means to convert raw data into information.

To get an idea of what these utilities can do, let's take a look at some of the tools in Microsoft Access—one of the first Windows-era database systems to provide built-in ways to load and transform data into something useful. Assume you work as the marketing manager for a company that produces breakfast cereals. You need not only to analyze competitive sales from another manufacturer but also to break down this analysis by individual brands.

You can certainly glean total sales information from publicly available documents, but you really want to try to break down competitive sales by individual brand. To do this, you might strike up an agreement with a major grocery store chain to get them to provide their sales information by brand in return for a small discount on your products. The grocery chain promises to send you a spreadsheet containing sales data from all its stores broken down by competitive brand for the previous year. The data you receive might look like Table 3.1.

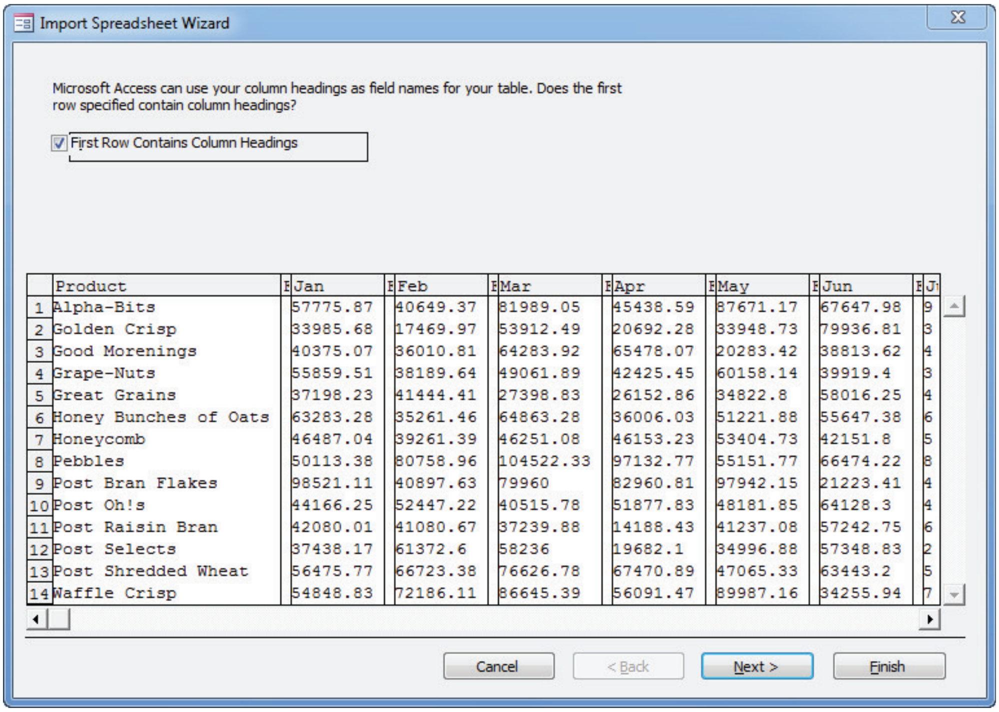
Table 3.1 Sample competitive sales data

Product		**Jan**		**Feb**		**Mar**
Alpha-Bits		57775.87		40649.37		. . .
Golden Crisp		33985.68		17469.97		. . .
Good Morenings		40375.07		36010.81		. . .
Grape-Nuts		55859.51		38189.64		. . .
Great Grains		37198.23		41444.41		. . .
Honey Bunches of Oats		63283.28		35261.46		. . .
. . . additional rows . . .						

It is clear that some blank columns that you do not need were added for readability. You also need to transform the data to end up with one row per product per month, and you have a separate table listing competitive products that has its own primary key, so you need to match on product name to get the key value to use as a foreign key.

Let's start by extracting the data from the spreadsheet into a more usable form. Microsoft Access can import data in many different formats, so let's fire up the Import tool to import a spreadsheet. In the first step, you identify the file and tell Access what you want to do with the output (import into a new table, append the data to an existing table, or link as a read-only table).

When you go to the next step, Access shows you a grid with a sample of the data it found, as shown in Figure 3.2. Because it determined that the first row might very well be usable as column names, it has used the names it found and has assigned generated names to the blank columns.



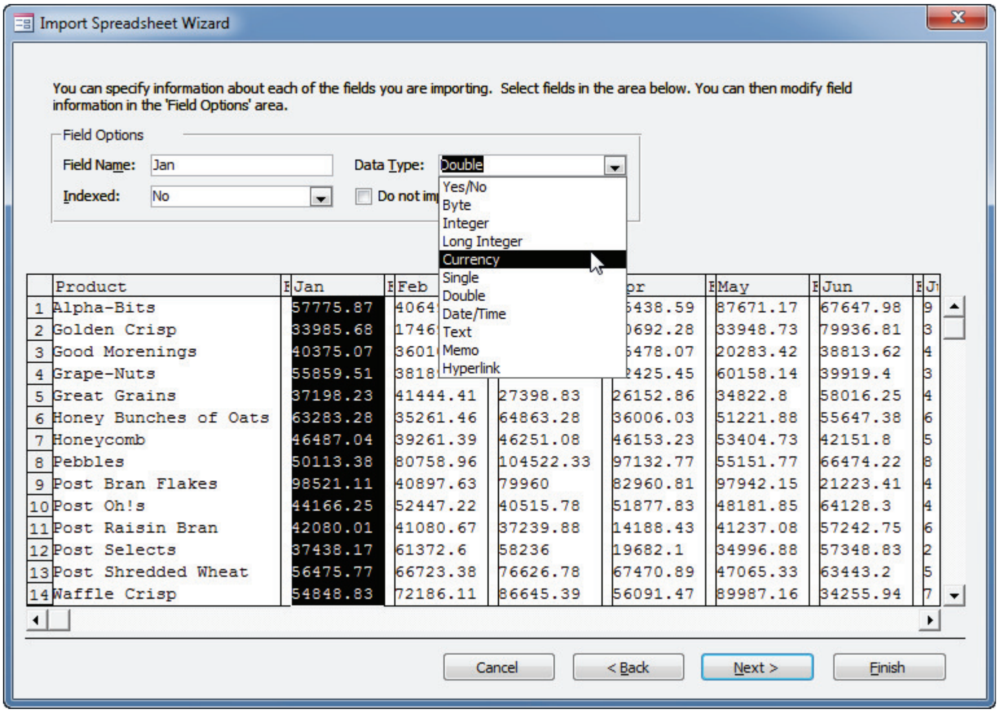
Microsoft Access can use your column headings as field names for your table. Does the first row specified contain column headings?

[✓] First Row Contains Column Headings

	Product	Jan	Feb	Mar	Apr	May	Jun	J
1	Alpha-Bits	57775.87	40649.37	81989.05	45438.59	87671.17	67647.98	9
2	Golden Crisp	33985.68	17469.97	53912.49	20692.28	33948.73	79936.81	3
3	Good Morenings	40375.07	36010.81	64283.92	65478.07	20283.42	38813.62	4
4	Grape-Nuts	55859.51	38189.64	49061.89	42425.45	60158.14	39919.4	3
5	Great Grains	37198.23	41444.41	27398.83	26152.86	34822.8	58016.25	4
6	Honey Bunches of Oats	63283.28	35261.46	64863.28	36006.03	51221.88	55647.38	6
7	Honeycomb	46487.04	39261.39	46251.08	46153.23	53404.73	42151.8	5
8	Pebbles	50113.38	80758.96	104522.33	97132.77	55151.77	66474.22	8
9	Post Bran Flakes	98521.11	40897.63	79960	82960.81	97942.15	21223.41	4
10	Post Oh!s	44166.25	52447.22	40515.78	51877.83	48181.85	64128.3	4
11	Post Raisin Bran	42080.01	41080.67	37239.88	14188.43	41237.08	57242.75	6
12	Post Selects	37438.17	61372.6	58236	19682.1	34996.88	57348.83	2
13	Post Shredded Wheat	56475.77	66723.38	76626.78	67470.89	47065.33	63443.2	5
14	Waffle Crisp	54848.83	72186.11	86645.39	56091.47	89987.16	34255.94	7

Figure 3.2 The Import Spreadsheet utility performing an initial analysis of the data

In the following step, Access shows you a display where you can select columns one at a time, tell Access to skip unimportant columns, and fix the data type that the utility has assumed. Figure 3.3 on the next page shows one of the data columns selected. The utility has assumed that the numbers, because they contain decimal points, should be imported as the very flexible `Double` data type. We know that these are all dollar sales figures, so it makes sense to change the data type to `Currency` to make it easier to work with the data. You can also see the "Do not import" check box (behind the drop-down) that you can select for columns that you want to ignore.


Import Spreadsheet Wizard

You can specify information about each of the fields you are importing. Select fields in the area below. You can then modify field information in the 'Field Options' area.

Field Options

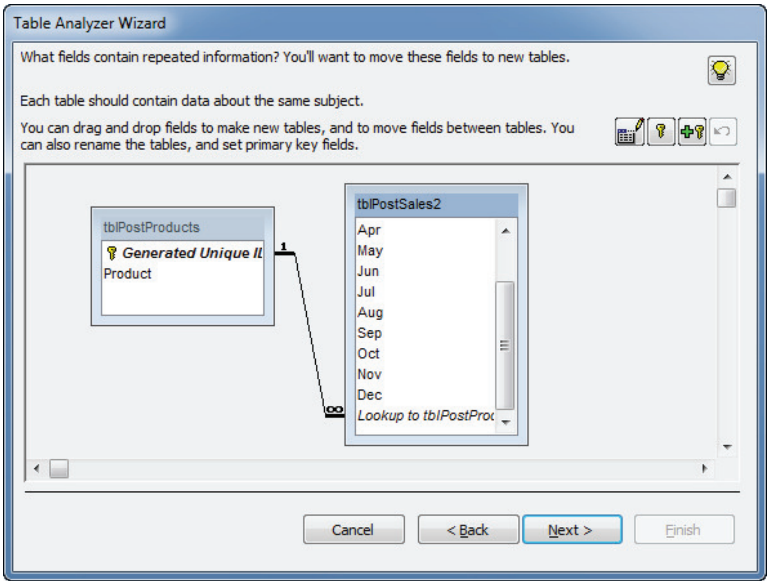
Field Name: Jan
Data Type: Double
Indexed: No
Do not im

	Product	Jan	Feb		pr	May	Jun	J
1	Alpha-Bits	57775.87	4064		5438.59	87671.17	67647.98	9
2	Golden Crisp	33985.68	1746		0692.28	33948.73	79936.81	3
3	Good Morenings	40375.07	3601		5478.07	20283.42	38813.62	4
4	Grape-Nuts	55859.51	3818		2425.45	60158.14	39919.4	3
5	Great Grains	37198.23	41444.41	27398.83	26152.86	34822.8	58016.25	4
6	Honey Bunches of Oats	63283.28	35261.46	64863.28	36006.03	51221.88	55647.38	6
7	Honeycomb	46487.04	39261.39	46251.08	46153.23	53404.73	42151.8	5
8	Pebbles	50113.38	80758.96	104522.33	97132.77	55151.77	66474.22	8
9	Post Bran Flakes	98521.11	40897.63	79960	82960.81	97942.15	21223.41	4
10	Post Oh!s	44166.25	52447.22	40515.78	51877.83	48181.85	64128.3	4
11	Post Raisin Bran	42080.01	41080.67	37239.88	14188.43	41237.08	57242.75	6
12	Post Selects	37438.17	61372.6	58236	19682.1	34996.88	57348.83	2
13	Post Shredded Wheat	56475.77	66723.38	76626.78	67470.89	47065.33	63443.2	5
14	Waffle Crisp	54848.83	72186.11	86645.39	56091.47	89987.16	34255.94	7

Figure 3.3 Selecting columns to skip and choosing a data type

The next step in the utility lets you pick a column to act as the primary key, ask the utility to generate an ID column with incrementing integers, or assign no primary key to the table. The final step allows you to name the table (the default is the name of the worksheet) and to invoke another utility after importing the table to perform further analysis and potentially reload the data into a more normalized table design. If you choose to run the Table Analyzer, Access presents you with a design tableau as shown in Figure 3.4. In the figure, we have already dragged and dropped the Product column into a separate table and named both tables. As you can see, the utility automatically generates a primary key in the product table and provides a matching foreign key in the sales data table.

Even after using the Table Analyzer, you can see that there is still plenty of work to do to further normalize the sales data into one row per month. You can "unpivot" the sales data by using a `UNION` query to turn the columns into rows, as shown in Listing 3.5. (See also Item 21, "Use `UNION` statements to 'unpivot' non-normalized data.")



What fields contain repeated information? You'll want to move these fields to new tables.

Each table should contain data about the same subject.

You can drag and drop fields to make new tables, and to move fields between tables. You can also rename the tables, and set primary key fields.

tblPostProducts
Generated Unique ID
Product

tblPostSales2
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Lookup to tblPostProd

Figure 3.4 Using the Table Analyzer to break out products into a separate table

Listing 3.5 Using a UNION query to "unpivot" a repeating group

```
SELECT '2015-01-01' AS SalesMonth, Product, Jan AS SalesAmt
FROM tblPostSales
UNION ALL
SELECT '2015-02-01' AS SalesMonth, Product, Feb AS SalesAmt
FROM tblPostSales
UNION ALL
  ... etc. for all 12 months.
```

The tools in Microsoft Access are fairly simple (for example, they cannot handle totals rows), but they give you an idea of the amount of work that can be saved when trying to perform ETL to load external data into your database. As noted earlier, most database systems provide similar—and in some cases more powerful—tools that you can use. Examples include Microsoft SQL Server Integration Services (SSIS), Oracle Data Integrator (ODI), and IBM InfoSphere DataStage. Commercial tools are available from vendors such as Informatica, SAP, and SAS, and you can also find a number of open-source tools available on the Web.

The main point here is that you should use those tools so that your data conforms to the data model that your business needs, not the other way around. A common mistake is to build tables that fit the incoming data as is and then use it directly in applications. The investment made to transform data will result in a database that is easy to understand and maintain in spite of the divergent data sources from which it may collect the raw input.

Things to Remember

- ETL tools allow you to import nonrelational data into your database with less effort.
- ETL tools help you reformat and rearrange imported data so that you can turn it into information.
- Most database systems offer some level of ETL tools, and there are also commercial tools available.

Item 20: Create Summary Tables and Maintain Them

We mentioned previously (in Item 18, "Use views to simplify what cannot be changed") that views can be used to simplify complex queries, and we even suggested that views can be used to provide summarizations. Depending on the volume of data, there are times when it may be more appropriate to create summary tables.

When you have a summary table, you can be sure that everything is in one place, making it easier to understand the data structure and quicker to return information.

One approach is to create a table that summarizes your data in your details table, and write triggers to update the summary table every time something changes in the details table. However, if your details table is frequently modified, this can be processor intensive.

Another approach is to use a stored procedure to refresh the summary table on a regular basis: delete all existing data rows and reinsert the summarized information.

DB2 has the concept of summary tables built into it. DB2 summary tables can maintain a summary of data in one or multiple tables. You have the option to have the summary refreshed every time the data in underlying table(s) changes, or you can refresh it manually. DB2 summary tables not only allow users to obtain results faster, but the optimizer can use the summary tables when user queries indirectly request information already summarized in the summary tables if `ENABLE QUERY OPTIMIZATION` is specified when you create the summary table. Although there may still be "costs" associated with all that