

# **Data-Driven Asset Management**

*An example for making any operation data driven*

Richard G. Lamb

**Chapter 4**  
**Super Tables from Operational Data**



This work is licensed by Richard G. Lamb under a [Creative Commons Attribution 4.0 International License \(CC BY\)](https://creativecommons.org/licenses/by/4.0/). Users are free to copy and redistribute the material in any medium or format and to remix, transform, and build upon the material for any purpose, even commercially. However, with the freedoms, users must give appropriate credit in a reasonable manner and there may not be legal terms or technological measures applied that legally restrict others from doing anything with the materials that the license permits.

Information contained in this work has been obtained from sources believed to be reliable. Neither the author nor publisher guarantee the accuracy or completeness of any information published herein and neither the author nor publisher shall be responsible for any errors, omissions, or damages arising out of the use of this information. The work is published with the understanding the author and publisher are supplying information, but not attempting to render professional legal, accounting, engineering or any other professional services. If such services are required, the assistance of an appropriate professional should be sought.

**Trademarks:** Microsoft, Microsoft Office, Excel Access, Oracle, SAP, Tableau, Power BI, Maximo and Track are registered trademarks in the United States and other countries.

## Contents

Chapter 4 Super Tables from Operational Data .....	1
4.1. Perspective .....	3
4.2. Scenarios for Demonstration .....	4
4.3. Case 1: Foundation Super Table .....	6
4.3.1. Identify, Extract and Import or Link .....	6
4.3.2. Translation Tables .....	8
4.3.3. Overview of the Query Process .....	10
4.3.4. Joins and Their Ramifications .....	13
4.3.5. Coding the Design Grid .....	17
4.3.6. Generate the Super Table .....	24
4.4. Case 2: Seek Outlier Work Orders .....	25
4.4.1. Measurement for Outliers .....	26
4.4.2. Activating Aggregation .....	27
4.4.3. Planning the Aggregation .....	29
4.4.4. Arriving at the Z-Score for Hours .....	30
4.4.5. Student's T-Score as Alternative .....	33
4.4.6. Expression and Where Aggregations .....	35
4.5. Case 3: Seek Outliers with Lead Craft .....	35
4.5.1. Planning the Lead Craft Aggregation .....	36
4.5.2. Arriving at Z-Score with Craft Lead .....	36
4.6. Comparison of Findings .....	43
4.7. SQL in the Background .....	44
4.8. Administration of Tables .....	45
Bibliography .....	46



## **Chapter 4**

### **Super Tables from Operational Data**

The definition of a data-driven asset management bears repeating. It is defined as using the firm's operational data to augment the experience and judgement of its operatives, managers, analysts and engineers as they plan, organize, conduct and control their processes.

The definition is repeated because this chapter will focus on two key words in the definition—operational data. Because data-driven asset management draws heavily on the firm's data, it must have the organizational ability to build super tables as the gateway to its operational data.

Let's establish a standard by which an organization would be judged as one with the organizational ability for super tables. For one, most role holders in the asset management operation would be conversant in super tables. Some role holders would have taken or been given the opportunity to acquire the skills to build super tables. Meanwhile, there has been a shrinking number of role holders who are neither conversant nor skilled in super tables but have the necessary basic skills to engage them in their assigned process tasks.

Furthermore, leadership would advocate for organizational development strategies to progressively increase the proportion of role holders who have the skills to build tables. As the proportion grows, the evolving effectiveness and efficiency of the asset management organization will have been observable. Just as importantly the personnel to the asset management enterprise would express an expectation for a brighter career.

The chapter serves two purposes toward becoming data driven. First, it is written for readers who are seeking to acquire the skills to develop super tables. As the training platform, the chapter will introduce, demonstrate and explain practices an SME reader would recognize as essential to topflight asset management. By emulating the practices with

## Chapter 4

their own data or data provided with the chapter, they will gain the skills to develop any envisioned super table.

Second, the chapter is written for leaders, operatives and consultants who recognize their need to become conversant. For them, remaining relevant requires that when an insight deliverable is being imagined and developed, they must be a full participant in its discussion.

A careful read will make them conversant. Alternately, they can assign the task to prepare and present a concise, clear explanation to a person who has become conversant or fully skilled. They, in turn, can use the chapter to form the presentation.

The example of the chapter is constructed with representative variables from a leading CMMS. The data of the example is contained in the file SupTbl\_Chap4.xlsx, available at [www.analytics4strategy.com](http://www.analytics4strategy.com).

Although representative, all identifying variables of the data have been either masked or redacted. The hour and dollar variables have been scrambled such that readers should not attempt to compare what is demonstrated to their own plants. In other words, the data is representative but no longer has a plant.

The chapter will first establish some initial perspective. It will then introduce a sequence of three cases as data-driven practices. For each case the chapter will demonstrate and explain how the data are built into an enabling super table. Especially notable of the cases is that few asset management organizations have the skills to conduct the demonstrated practices even though they are fundamental and essential to asset management. For them, the chapter will be an immediate step into the new age.

The reader will achieve full competency from the chapter. However, the chapter will not cover every issue and element. After reading the chapter, readers are encouraged to read Chapters 8 through 13 of the text, “Access 2016 Bible. The comprehensive tutorial resource.” Furthermore, every feature and function introduced in the chapter and the text can be viewed in a several-minute YouTube video.

Because the variables are common to any CMMS, readers are encouraged to emulate with their own data what is demonstrated—use the

chapter as a go-by. It is invariable that people who begin with a go-by, immediately find themselves ranging out to build newly imagined super tables.

### 4.1. Perspective

Chapter 2 introduced what we are trying to do. Figure 2-8 showed that we query for standard reports from our systems. They collectively contain the variables we need to build our intended insight deliverables. However, we are divided and conquered by our systems unless we can join the variables in a single super table. This section will establish some basic perspectives of super tables. We begin with three truths.

First, almost all modern operating systems allow their data to be extracted in table format as standard reports. The reports are formatted as columns (variables) and rows (records or cases). Columns are titled as headers. Rows are never titled. Nor are there empty or summary rows.

Second, tables from any one or more systems can be joined in a single super table. The only requirement is that they must have a common identifying variable with which to join them.

Third, bad data is rarely a deal-killer. We are rarely dependent upon one variable to get an insight. Furthermore, cleansing will often neutralize the flaws.

A final perspective. The process of building super tables entails two stages—foundational and aggregational.

The first stage builds the foundational table to one or more insights. The stage joins source sub tables in a massive raw table of all source variables and then pulls the variables we want into the intended super table. Additional variables to the super table are typically formed or calculated upon the source variables.

The second stage designs aggregation variables into super tables. The concept is to specify groups in the foundational table. In turn, aggregation variables are created for the groups. They can be count, sum, average, standard deviation, variance, min-max and first-last.

## Chapter 4

### 4.2. Scenarios for Demonstration

The scenarios of the Chapter, as explanation by demonstration, will develop a super table and two aggregations for the purpose of gaining insight to craft hours to a report month. The hours will be narrowed to those that were engaged by the maintenance tasks of the work orders that were completed during month.

The scenarios are a consequential topic with which to explain super tables. Craft hours are the largest and most controllable expense of asset management. As data, they are also the life blood to all sorts of insights that are not available from a CMMS.

We will use three standard reports that are natural and typical to any CMMS. First are work orders. Second are the maintenance tasks to each order. Third are the hours per individual craft to the maintenance tasks that engaged them.

Some standard reports to a CMMS offer a considerable list of variables for selection. This is typical to the standard report of work orders. The three tables were extracted from the CMMS with not only the variables we know we want to employ in the planned insight deliverable but also with those we may find ourselves wishing we had included as the deliverable takes form.

The chapter will work through a sequence of three cases. Each is an extension upon Case 1 to build a foundation super table with the three standard sub tables from the CMMS.

Cases 1 and 2 extend from the super table to demonstrate an important, fundamental practice for asset management. Some operations evaluate a month's work orders based upon exploring the top ten in cost. However, the problem is that the top ten may legitimately be the top ten rather than indicative of something of interest.

Case 2 of the scenario will search for true outliers. It will use a statistical Z-score standardized method. As a result, we can ask ourselves which orders are beyond some percentage of occurrence such as 95 percent. The analysis will seek outliers for investigation among groups as permutations of cost center and maintenance type.



## **Super Tables from Operational Data**

Case 3 will take an important further step for seeking out true outliers. The purpose is to conduct a more realistic search. The case works around a debilitating weakness that is inherent to how the representative CMMS is configured.

The configuration classifies orders by type (e.g., preventive, on-condition, corrective) but not by craft lead (e.g., machinist, mechanical, electrical, instrumentation). The distinction is difficult because more than one craft is typically engaged in an order.

Therefore, Case 3 will classify records in a way the representative CMMS does not. It will seek outliers among groups as permutations of cost center, maintenance type and lead craft.

Although not a case, there will be a final demonstration. The demonstration will compare the findings of the two cases to the top-ten method and to each other.

The chapter will follow the steps introduced in Chapter 2 and flowcharted in Figure 2-9. Recall they are as follows:

1. Identify relevant tables.
2. Extract tables from source systems.
3. Import or connect source tables into Access.
4. Build foundation super table.
5. Explore and cleanse the data.
6. Build aggregate variables into the super table.
7. Administer, update, upgrade, and disseminate super table.

This chapter will speak to the first four steps to identify, extract, import or connect and build the foundation super table, and the sixth step to build aggregate variables to the super table. The explore and cleanse step will be a topic of Chapter 6 explaining how to get our data and maintenance process under control. Finally, with respect to the seventh step, the chapter will speak to some organizing issues within the Access file.

## Chapter 4

### 4.3. Case 1: Foundation Super Table

We begin by building the foundation super table. Along the way, the section will explain the process, and the principles and practices of building tables. The data to the explanation by demonstration are available for download at <https://analytics4strategy.com/ddassetmgt> as the file SupTbl\_Chap4.xlsx.

#### 4.3.1. Identify, Extract and Import or Link

As the first step, we have found that all pertinent data to our envisioned super table reside in three tables to the CMMS. They are work orders, tasks to the orders and craft hours to the tasks.

Operating systems typically give us a choice to download to Excel, Access and other destinations. The demonstration will take the trail through Excel.

The next step is to download the identified source system tables to an Excel file. I recommend placing them in a single file. As a staging area, we may want to conduct basic preparations of the data. We may want to change headers. We may want to conduct some know-thy-data activities. Of course, we can conduct the same work in Access, but some tasks may be easier in Excel.

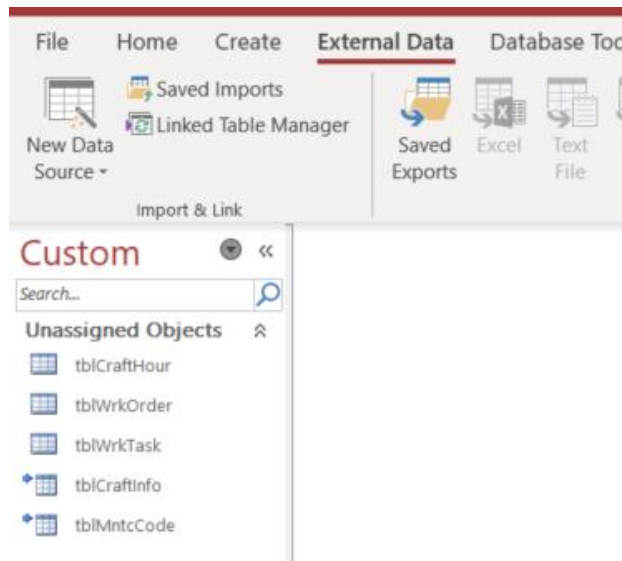
The third step of the process is to pull the extracted tables into Access. Once again, a simple step. We merely follow pop-up windows. Start the process by clicking the New Data Source icon on the Access Ribbon under the External Data heading of the menu.

We will be given four choices—from file, data base, other services and online services. Given that we have chosen to download our tables to Excel from the CMMS, we would now select the from file option and select Excel. If we had downloaded the system tables to another Access file, we would have selected the from Database option and selected Access.

Figure 4-1 shows another important distinction for managing data-driven processes and insights. Notice the list of tables. There are two additional tables to the three we previously identified and extracted from the CMMS. They will be explained later in the chapter.

## Super Tables from Operational Data

At this point, an important point to notice is the symbol to the left of the bottom-two tables. This means that the tables have not been imported. Instead, they are linked to the source—in this case the Excel file at which they are managed. Each time the query is run, it will reach outside for the latest version of the data. The option to import or link is made available as we follow the process beginning with selecting a source.



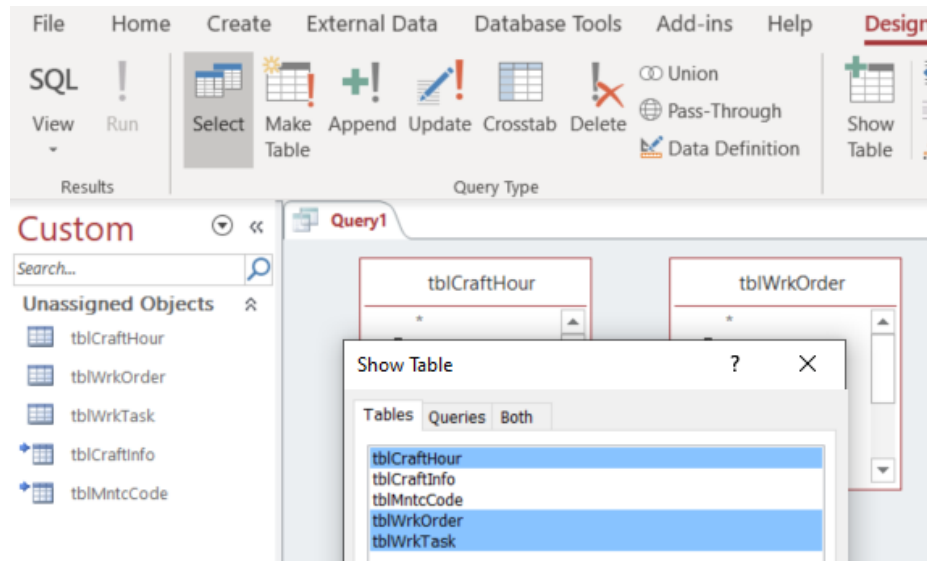
**Figure 4-1: Three downloaded tables and two clarifying tables.**

The difference is noteworthy because the seventh step is to administer sub tables and super tables. The users may link rather than import some or all tables so to be assured that they have the latest edition of a done-by-one, used-by-all source. If we import data and it is later updated or kept clean elsewhere, our insights may be misinformation unless they are linked the done-by-one edition.

The purpose of the two linked tables and how they are developed is explained in the next section. As external to the CMMS, they are what the book calls translation tables.

Now that the tables are loaded, we select the Create option in the menu and then the Query Design icon in the Queries section of the ribbon. As shown in Figure 4-2, the Select query will open automatically with the Show Table pop-up window.

## Chapter 4



**Figure 4-2: Tables imported and linked to Access for building super tables.**

Notice that there are six types of queries. Most work is with select queries. The others have purposes supplemental to the select query. The query types will be explained as the chapter unfolds.

The pop-up shows the five tables. We can highlight the tables we want in the super table and click Add (not shown in the figure). Alternately, we can drag the tables listed in the left-most view of available objects into the work area.

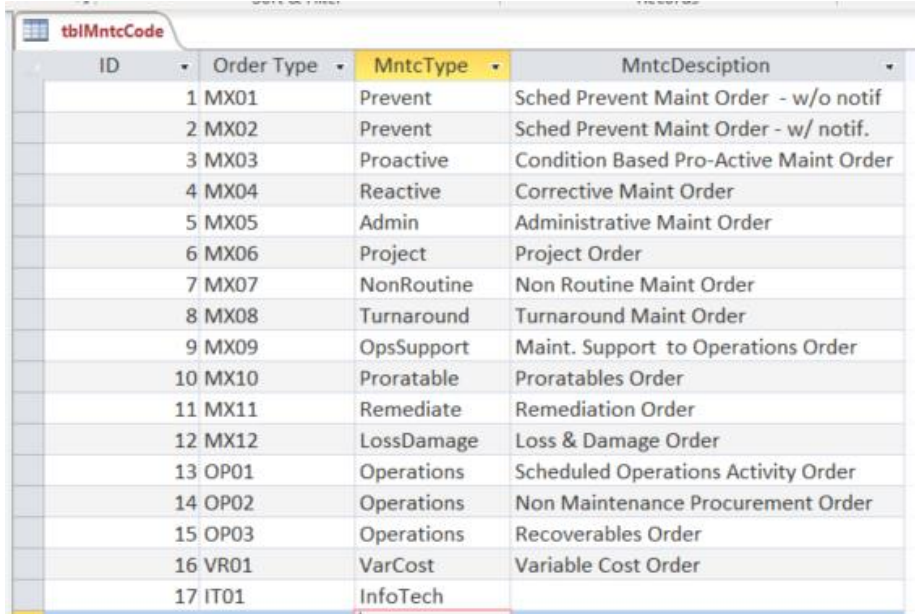
Another point, we are not restricted to table objects. We can also pull queries into a query.

### 4.3.2. Translation Tables

Two of the five tables in the object view are what the book calls translation tables. They can be used to clarify variables in the final super table and, thus, insights delivered from it. They can be used to cleanse data. They can be used to create uniformity between organizational entities. I have used them to join the data of two CMMS generations, when the supplanting system would have otherwise left the organization without contiguous historical data.

## Super Tables from Operational Data

It is an easy, down-dirty method. To explain the concept, Figure 4-3 shows the table, tblMntcCode, to translate arcane order type code to plain English.



ID	Order Type	MntcType	MntcDescription
1	MX01	Prevent	Sched Prevent Maint Order - w/o notif
2	MX02	Prevent	Sched Prevent Maint Order - w/ notif.
3	MX03	Proactive	Condition Based Pro-Active Maint Order
4	MX04	Reactive	Corrective Maint Order
5	MX05	Admin	Administrative Maint Order
6	MX06	Project	Project Order
7	MX07	NonRoutine	Non Routine Maint Order
8	MX08	Turnaround	Turnaround Maint Order
9	MX09	OpsSupport	Maint. Support to Operations Order
10	MX10	Proratable	Proratables Order
11	MX11	Remediate	Remediation Order
12	MX12	LossDamage	Loss & Damage Order
13	OP01	Operations	Scheduled Operations Activity Order
14	OP02	Operations	Non Maintenance Procurement Order
15	OP03	Operations	Recoverables Order
16	VR01	VarCost	Variable Cost Order
17	IT01	InfoTech	

**Figure 4-3: Translation table for order type.**

Translation tables can be built and updated inside Access or elsewhere such as Excel. The explanation will assume the Excel strategy. To build, update and upgrade them in Access, the readers are referred to Chapter 3 of Alexander and Kusleika. Once built, they can be maintained and made available to any super table.

The Excel process begins by building a list of all existing categories in Excel. The list is usually taken from the operating system or systems.

In the figure we have extracted the variable of order type code from the CMMS. Thence, we have created the two variables MntcType and MntcDescription as alternatives to the OrderType code. From Access, we can either import or link to the translation table.

## Chapter 4

It follows that we can return at any time to the translation table and add clarifying variables. An example would be a variable based upon the distinctions of the task types for reliability-centered maintenance.

The other translation table in the object pane allows the super table to classify craft types by useful categories. With the translation table, tblCraftInfo, the hours by individual crafts are translated to different designations rather forcing us to work with the many fragmented but equivalent titles in the CMMS. By the table, 42 craft titles have been translated to 8 categories.

Over time, expect that an entity will develop and maintain a collection of translation tables. They will be called into existing and developing super tables.

Accordingly, think in terms of storing the translation tables as a collection in a single Access file, place them for access by anyone and manage them as a role in the data-driven asset management operation.

When we run a query linked to the translation tables, the latest version is pulled into the run. Consequently, users whose role entails tasks to update insight deliverables can know that the source data is up to date by virtue of being linked to the administered source.

### 4.3.3. Overview of the Query Process

The ribbon under the tab, Design, reveals that there are six types of queries. However, most of everything is done in a select query. The others are supplemental tools to build, update and administer the select query. How they play will be explained opportunistically as this and later chapters unfold.

The definitions of the six types of queries (shown in Figure 4-2) are as follows:

- **Select:** Builds the super table from one or more subtables. Aggregation is constructed in the select query.
- **Make Table:** Converts a select query to a “hard” table.
- **Append:** Adds rows of data to an existing table.
- **Update:** Changes rows to variables in a table or query.
- **Delete:** Removes rows to variables in a table or query.

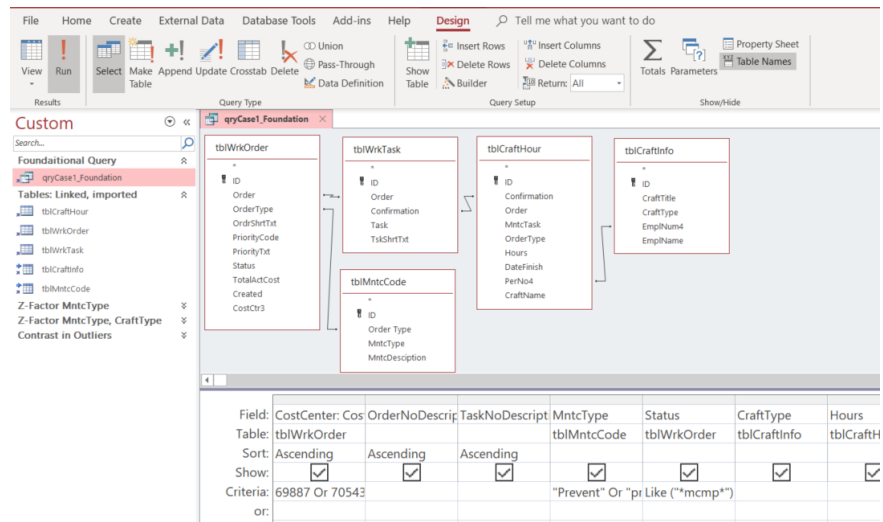
## Super Tables from Operational Data

- **Crosstab:** Makes long tables wide—e.g., a variable of months transformed to a table with variables for each month.

Our first intent is to open a new select query. Click Create in the menu bar and then Query Design in the Queries section of the ribbon. Access will open a new select query. Right click the query's name tab, select save and give the query its name (qryCase1\_Foundation).

Next, we drag the tables we want into the query and join them. Thence, we drag the variables we want into the design grid. In the grid, we mold the variables to our super table.

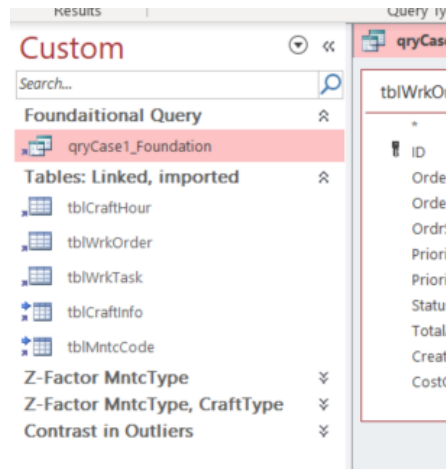
Figure 4-4 shows the layout of the design view. We can see how the pieces come together. To get a deeper look, the sections of the layout will be enlarged and explained in the paragraphs to come.



**Figure 4-4: Design view to the super table.**

Figure 4-5 shows what geeks call objects—tables, queries, etc. On inspection of the objects view, we can see the original five source tables. However, other objects emerged as the queries of the chapter were developed. The section, Administration of Tables, will explain the objects view and how objects are organized as shown in the figure.

## Chapter 4



**Figure 4-5: List of objects associated with the super table.**

This is an opportunity to recommend a good practice. Notice the “tbl” prefix to table objects and “qry” prefix to the query object. In a list of objects, the prefixes allow us to readily distinguish one type of object from another. The value of the practice looms large when there are many objects in an Access file.

Now let’s look at how the sub tables are joined in the super table. Figure 4-6 shows that all five tables have been pulled into the query. However, let’s note again that queries can also be pulled into a query. We are not limited to tables. This will be seen to happen for cases 2 and 3, and the comparison analysis.

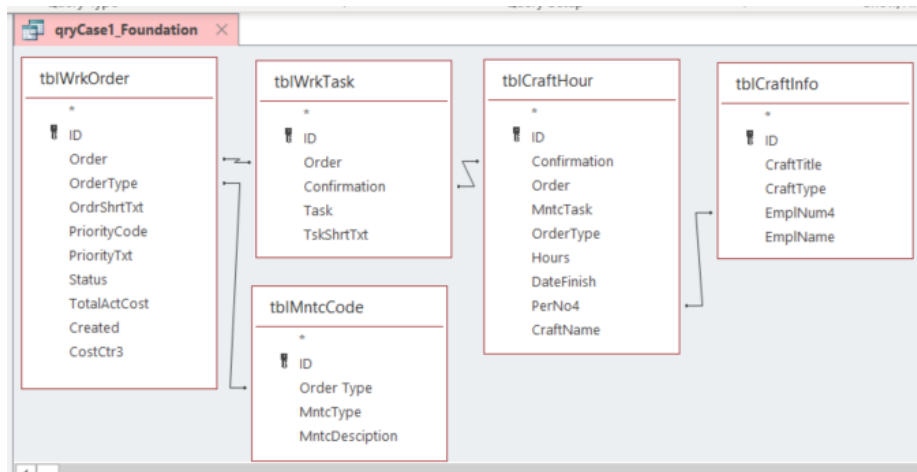
Recall that tables are joined by an identifier variable they have in common. Upon inspection we can see the tblWrkOrder and tblWrkTask tables are joined on order number, and the tblWrkTask and tblCraftHour tables are joined on confirmation number.

The previously explained two translation tables are also engaged. The tblWrkOrder and tblMntcCode tables are joined on the order type variable. The tblCraftHour and tblCraftInfo tables are joined on their respective employee identifier variables.

The last join demonstrates an important point. The joining variables need not be of the same name. They must only be of the same type—numeric or character.



## Super Tables from Operational Data



**Figure 4-6: All five tables to the super table and their joins.**

Joins between tables have types. They are classified as inner, left, right and outer. The types and ramifications of each will be explained in the next section.

Figure 4-4 showed the design grid at the bottom of the design view. It is the area that the joined tables are molded into our envisioned super table. What we see in the grid is actually the code that will return the super table.

Let's establish a reference point. Standard Query Language (SQL) is the universal code to pull data from one or multiple relational databases and build tables. However, the intent of Access is that the code we place in the design grid has become normal to our daily tasks.

When we are placing what we have become friendly with in the grid, the SQL code to actually do the deed is forming behind the curtain. This is called query by example (QBE). As we work in the grid, we are telling Access what we want to be coded as SQL. Therefore, we do not need to learn SQL because the QBE functionality does it for us.

### 4.3.4. Joins and Their Ramifications

Figure 4-6 showed the tables as joined by the line between a common variable to pairs of objects. However, there is more to it. As mentioned earlier, the join line represents one of four types of joins. We

## Chapter 4

will use Figure 4-7 to explain them. Then we will see how to make the choices for our super table.

Subtables					
TableA	TableB	Inner Join: Both tables have populated the case variable	TableA	TableB	Right Join: Opposite to left join.
Order#	Order#		Order#	Order#	
100126	100126	Left Join: All cases of Table A returned along with the populated and unpopulated cases of Table B.	100126	100126	100126
100236	100313		100726	100726	100313
100726	100726				100726
100810					
TableA	TableB	Left Join: All cases of Table A returned along with the populated and unpopulated cases of Table B.	TableA	TableB	Outer-join: Entirety of Tables A and B.
Order#	Order#		Order#	Order#	
100126	100126	Right Join: Opposite to left join.	100126	100126	100126
100236			100236		100236
100726	100726		100726	100726	100726
100810			100810		100810
					100313

Figure 4-7: Tables can be joined to return four different outcomes.

To the left in the figure are two tables—TableA and TableB—with order number as the common identifier variable. However, the cell contents are not one for one. Consequently, the join we chose will return different super tables as shown at the right of the figure.

The inner join will return a table in which the records are the ones in which both cells are populated with identical order numbers. Only the two such records are returned. All others do not appear in the super table.

The left join returns all populated cells from the left table. The non-matching cells in the right table appear as empty cells. In contrast, the right join will return the opposite outcome to the left join.

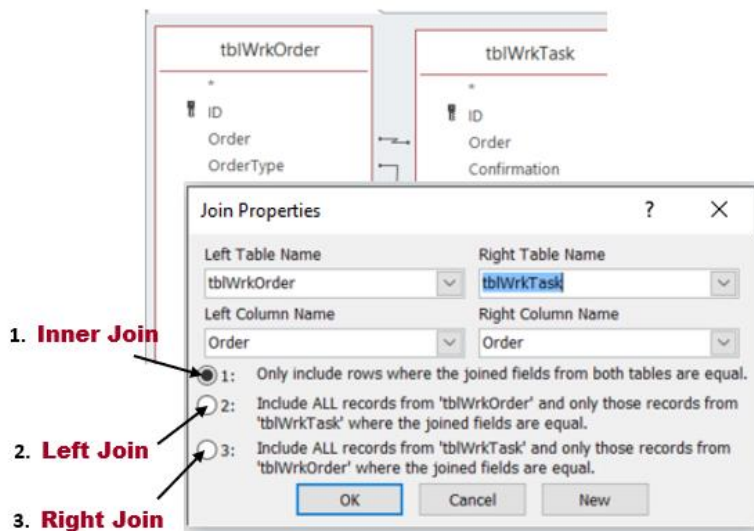
The point of the left and right join is that one table is the basis of the other. When working with them, it is good practice to confirm we are getting what we want rather than the opposite. We make the confirmation by running and inspecting the returned super table.

The nature of the outer join is apparent in the figure. Notice that all cases are returned. Non-matched cases to both tables are shown as empty (null) cells.

## Super Tables from Operational Data

Joins have ramifications for know-thy-data, audit and control. This is because our attention is drawn to a simple question. Why are there empty cells? When building and molding a super table it is good practice to vary the joins and look for nulls in the returned table. We often discover something.

Figure 4-8 shows how we set the joins between tables with Access. The process begins with a right-click on a join line in Figure 4-6 and selecting the Join Properties option. The Join Properties window shown in Figure 4-8 will pop up for specifying the join we want.



**Figure 4-8: Select the join type, if not the default inner join.**

In the figure, notice that the first option is an inner join. It is also the default. It is not necessary to work in the window if the intended join is an inner join.

In the pop-up, notice that Access regards tblWrkOrder as the left object to the join and tblWrkTask as the right object. It shows that the joining variables are the order number variable in each.

If we want a join other than the inner join, we select one of the two remaining alternatives—left and right. However, notice there is no option for an outer join. The absence is one of the few disappointments of Access.

## Chapter 4

However, there is an easy work around for creating an outer join. Do a left or right join. Thence, append it to the empty (null) variable rows to the opposite join. We would create an outer join table, pull it into the query and join the tables through it. A later chapter will demonstrate the power of outer joins to analyze work attainment.

The joins to the super table, qryCase1\_Foundation, are provided in Table 4-1. They are presented as they would appear in the Join Properties pop-up window (Figure 4-8).

**Table 4-1: qryCase1\_Foundation—joins.**

Side:	Left	Right
Table:	tblWrkOrder	tblWrkTask
Variable:	Order	Order
Join:	Inner	
Side:	Left	Right
Table:	tblWrkTask	tblCratHour
Variable:	Confirmation	Confirmation
Join:	Inner	
Side:	Left	Right
Table:	tblCratHour	tblCraftInfo
Variable:	PerNo4	EmpNum4
Join:	Inner	
Side:	Left	Right
Table:	TblWrkOrder	tblMntcCode
Variable:	Order Type	Order Type
Join:	Inner	

This is a good place to introduce the Append query. An append query is a simple matter. Rather than present a full discussion, the reader is advised to find a several-minute YouTube demonstration by browsing the internet for “append query access.”

In this section, the Append query is introduced as a tool to build an outer join. More commonly, it will arise when we are updating tables in Access rather than in Excel. It is usually more practical to append updates in Access. Another reason is that a table, periodically appended in

## Super Tables from Operational Data

Excel, may eventually grow to hold more records than Excel has the capacity to hold.

### 4.3.5. Coding the Design Grid

It is a choice of personal style. When extracting each table from its source system, consider selecting all potentially useful variables rather than only the ones for the currently developing super table. As we build our super table or use the data for other super tables, we are often glad we did.

Now we select the variables of interest to the super table named qryCase1\_Foundation. This is done by dragging each variable from the design upper view into the Field row of the design grid. In most cases the Table row will populate automatically. Thence, four other aspects remain to be coded as it is necessary—Sort, Show, Criteria and Or.

The code as placed in the design grid of the super table is provided in Table 4-2. The code will be explained by the paragraphs to come. The alternatives to the shown code will also be provided along the way. Therefore, the reader will be introduced to almost every code they will ever need for any envisioned insight.

**Table 4-2: qryCase1\_Foundation—grid code.**

<b>Join</b>	See Table 4-1		
<b>Field</b>	CostCenter: Cost-Ctr3	OrderNoDescription: [tblWrkOrder].[Order]&": "&[OrdrShrtTxt]	TaskNoDescription: [Task]&": "&[TskShrtTxt]
<b>Table</b>	tblWrkOrder		
<b>Sort</b>	Ascending	Ascending	Ascending
<b>Show</b>	Y	Y	Y
<b>Criteria</b>	69887 Or 70543 Or 70107 Or 69839		
<b>Or</b>			
<b>Field</b>	MntcType	Status	CraftType
<b>Table</b>	tblMntcCode	tblWrkOrder	tblCraftInfo
<b>Sort</b>			
<b>Show</b>	Y	Y	Y

## Chapter 4

<b>Criteria</b>	“Prevent” or “Proactive” or “Reactive”	Like(“*MCMP*”)	
<b>Or</b>			
<b>Field</b>	Hours	DaysPastCreated: [DateFinish]-[Created]	Created
<b>Table</b>	tblCraftHour		tblWrkOrder
<b>Sort</b>			
<b>Show</b>	Y	Y	Y
<b>Criteria</b>			Between #1/1/2011# And #2/27/2012#
<b>Or</b>			
<b>Field</b>	Order	Confirmation	
<b>Table</b>	tblWrkOrder	tblWrkTask	
<b>Sort</b>			
<b>Show</b>	Y	Y	
<b>Criteria</b>			
<b>Or</b>			

The first variable identifies the cost center. The variable CostCtr3 was dragged down and its source table, tblWrkOrder, automatically populated the Table row of the grid.

However, we want to give the variable clarity in the super table. We prefer CostCenter as the named variable. Accordingly, we rename it—give it an alias—by placing CostCenter: (name, colon and space) in the Field row of the grid.

Not all variables in the grid may need to appear in the returned super table. Accordingly, the Show row allows us to cause a variable to not appear although it is in the grid. The default is to appear. Otherwise, we would deactivate the check box (depicted as Y/N in Table 4-2) to the variable.

The Sort row commands the table to be returned with the variables as ascend, descend or no order if empty. In the code, the table will be sorted in ascending order from left to right—cost center, orders to cost center and tasks to order.

## Super Tables from Operational Data

In the Criteria row, we mold the table by filtering the records to each variable. With respect to CostCenter we have reduced our table to 4 of the plant's 95 cost centers.

Notice the Or operator in the code of the Criteria row. The operator indicates that we want all the records to the four centers—all else excluded. An And operator in the variable (column) makes no sense. It would mean we want records that are a combination of the four cost centers. We would get a table with no records.

Table 4-3 provides a list of criteria expressions for all data types. Connecting back to the Or operator, if we want to apply more than one criteria expression to a variable, we would insert the Or operator between each expression.

**Table 4-3: Criteria expressions for all data types.**

Criteria name	Code	Effect
Equals	"x" for text; x for numeric.	Searches for values equal to x
Does Not Equal	Not in ("x") for text. Not in (x) for numeric.	Searches for all values except those equal to x
Null	Is Null	Searches for empty fields
Not Null	Is Not Null	Searches for non-empty fields.

The And condition exists between the columns of the design grid. Collectively, the criteria across the grid define the records to remain in the super table. Figure 4-4 and Table 4-2 showed the criteria that have been coded for 4 of the 11 variables. In the super table, they collectively set filters for cost center, maintenance type, status and dates created.

Although not necessary for Case 1, what if we want an Or case between variables in the grid? The answer is the Or row of the grid. We can create groups with different And criteria between the variables. To do it, each group would be coded in a separate Or row.

The next two variables in Table 4-2—OrderNoDescription and TaskNoDescription—return a combination of variables as a single variable. Additionally, the code demonstrates some other important basics to building and molding super tables.

Let's look at the first of the two variables. The purpose of the code is to combine the order number and text description variables in a single

## Chapter 4

variable. An example is the order and description variables coded to return a cell such as 6000937432: Pump1871 seal leaking.

Notice the piece of code to the variable—[tblWrkOrder].[Order]. The code is necessary because a variable named Order occurs in more than one sub table. When dragged into the grid, Access will tell us that it is not able to know which one we want until we provide additional information. The additional information is the source table as identified within the first pair of square brackets. The variable is identified within the second pair. Finally, to make it work, notice that ‘.’ (dot) is placed between the table and field brackets.

Next notice the & operator. The geek word for it is the concatenation operator. By placing the operator between two variables, they are returned as a single (concatenated) variable.

However, in this case we want some punctuation and spacing to make the variable more readable. This is done with the code—“: “ (colon space)—between the concatenation operators. The parentheses are required because the colon and space are text rather than numeric. Note that it is typical to all codes that text be bracketed with parentheses and the numbers be absent of parentheses.

Now let’s look at the next three variables in the grid of Figure 4-4 and Table 4-2—MntcType, Status and CraftType. All three are simple drags. MntcType includes a criteria using the Or operator. The code causes the table to return 3 types of maintenance from the 17 categories that are inherent to the data extracted from the CMMS.

Notice the parenthesis bookends to the desired types using the Or operator. Except the parentheses, this is the same pattern as seen previously for the CostCenter criteria. This is because the maintenance type variable is text rather than numeric. In contrast, cost center is a variable of numeric values.

Next notice the code—Like(“\*MCMP\*”)—as the criteria to the Status variable. It is called the Contains criteria. We could have coded the Equals criteria—“MCMP”—that was included in Table 4-3.

However, upon inspecting the data, some of the status cells to the variable were found to be populated with multiple statuses. The Equals



## Super Tables from Operational Data

criteria would not include them in a table, but the Contains criteria would.

Among the raw data, the pattern was observed as occasionally including MCMP. Although a check of the forming super tables did not find the pattern, we want a criteria to recognize the pattern if it should arise in future monthly updates.

Table 4-4 provides the list of available text criteria. We all know of many or all of them from our experience with Excel and other software.

**Table 4-4: Criteria for working with text strings.**

Criteria name	Code	Effect
Contains	Like ("*x*")	Searches for all values that contain x
Does Not Contain	Not like ("*x*")	Searches for all values that do not contain x
Begins With	Like ("x*")	Searches for all values beginning with x
Ends With	Like ("*x")	Searches for all values ending with x
Comes After	>= "x"	Searches for all values that come before x in alphabetical order
Comes Before	<= "x"	Searches for all values that come after x in alphabetical order

The next three variables in the design grid are Hours, DaysPastCreated and Created. Two additional and commonly occurring codes can be spotted in the design grid. One demonstrates the creation of calculated variables. The other is to specify date ranges.

The code in the Field row—DaysPastCreated: [DateFinish]-[Created]—is a calculation we have given the name DaysPastCreated. The big point is that any calculation is possible—simple or complex. There is no great magic. The expressions of our Excel experience and experience with other software will often transfer. If what we are attempting upon our experience does not work, we can query the internet for a solution—query for “access expressions.”

Even though they have not been called for the super table, two expressions will be introduced here because we find ourselves frequently

## Chapter 4

using the first of them. They are the conditional expressions—IIF() and Switch(). They are coded as shown in Table 4-5.

**Table 4-5: The conditional expressions IIF() and Switch().**

Function name	Code	Effect
If then	IIF(logical test, value if true, value if false)	Evaluates a specific condition and specify results whether the condition meets True or False values
Switch	Switch( logical test1, value1, logical test2, value2, ... logical test_n, value_n )	Evaluates a list of paired expressions and returns a value or an expression associated with the first expression in the list that is True

It is apparent from the table that the first IIF(), is familiar to most of us. Alternately, Switch() allows us to avoid IIF() expressions that get messy when we want to nest an IIF() within an IIF() and so on.

For the variable, Created, we are working with dates. Notice that # (pound) bookends the dates. This differs from the bookends for text and numeric variables. The expression calls for all records within and including the dates.

This is a good place to introduce the body of criteria for bounding date and numeric variables. Table 4-6: shows them for date variables and Table 4-7 for numeric variables.

**Table 4-6: Criteria for working with date variables.**

Criteria name	Code	Effect
Between	Between “#mm/dd/yyyy#” and “#mm/dd/yyyy#”	Searches for dates that fall between the specified dates
Before	< “#mm/dd/yyyy#”	Searches for dates before a certain date
After	> “#mm/dd/yyyy#”	Searches for dates after a certain date
Today	=Date()	Searches for all records containing today’s date
x Days Before Today	<=Date()-x	Searches for all records containing dates x or more days in the past

## Super Tables from Operational Data

**Table 4-7: Criteria for bounding numeric variables.**

Criteria name	Code	Effect
Between	Between x and y	Searches for all values in the range between x and y
Less Than	< x	Searches for all values smaller than x
Less Than or Equal To	<=x	Searches for all values smaller than or equal to x
Greater Than	> x	Searches for all values larger than x
Greater Than or Equal To	>=x	Searches for all values larger than or equal to x

Now for the final two variables to the super table. They are Order and Confirmation.

Including the two variables in the super table is a proactive practice rather than necessary for the foundation super table. As we work along to build the super table, we may find that we need them.

Another reason to include them is that we may find ourselves building other super tables with the foundation super table. It is very likely that the need for the inherent identifiers will arise.

With qryCase1\_Foundation, we could join on the concatenated variable. Or we could tease the order number out of the concatenated variable with string functions. However, it is good practice to bring along the untouched inherent identifiers.

In the practice of building super tables with Access, we will find that the demonstrated and additional referenced code in this section covers most of everything we would ever call for. We will also recognize that most of what has been introduced matches or is very similar to what we know from working with Excel spreadsheets and other software.

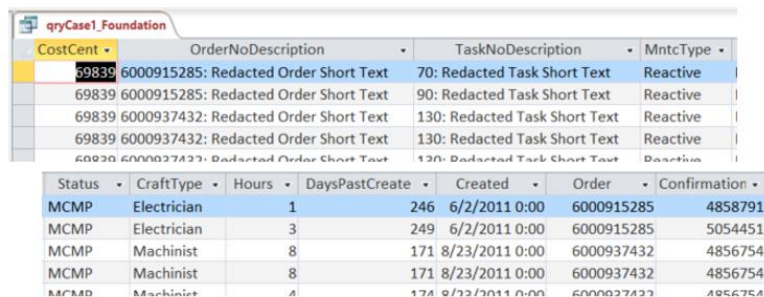
This brings us back to a point made earlier. Building super tables is more a matter of creativity with an insight deliverable in mind than it is to be a data geek. In fact, the skills of data science have little meaning to us normal people working with our data to reach an insight. We can function quite nicely and still be largely ignorant of data science beyond what we already know as modern role holders.

## Chapter 4

### 4.3.6. Generate the Super Table

Now to generate the foundation super table. However, rather than a final event, we should flip between the design and table views as we develop the table. One reason is to confirm, at each step, that we are getting what our code was intended to give us. Another reason is that we will often find ourselves following discoveries that take us to new insights and revelations.

Notice the table icon at the far left of the ribbon shown in Figure 4-4. It returns the super table as shown in Figure 4-9. The figure cuts the table in two pieces and shows only the first 4 of its 687 rows. Notice the variables to the table are as we named them.



CostCent	OrderNoDescription	TaskNoDescription	MntcType
69839	6000915285: Redacted Order Short Text	70: Redacted Task Short Text	Reactive
69839	6000915285: Redacted Order Short Text	90: Redacted Task Short Text	Reactive
69839	6000937432: Redacted Order Short Text	130: Redacted Task Short Text	Reactive
69839	6000937432: Redacted Order Short Text	130: Redacted Task Short Text	Reactive
60030	6000937432: Redacted Order Short Text	130: Redacted Task Short Text	Reactive

Status	CraftType	Hours	DaysPastCreate	Created	Order	Confirmation
MCMP	Electrician	1	246	6/2/2011 0:00	6000915285	4858791
MCMP	Electrician	3	249	6/2/2011 0:00	6000915285	5054451
MCMP	Machinist	8	171	8/23/2011 0:00	6000937432	4856754
MCMP	Machinist	8	171	8/23/2011 0:00	6000937432	4856754
MCMP	Machinist	8	171	8/23/2011 0:00	6000937432	4856754

**Table 4-9: The table returned from the code.**

This is a good place to introduce a functionality with which to confirm and explore the table. Notice the Totals option in the Records section of the ribbon (not shown in the figure). When selected, a Totals row will appear at the bottom of the table view. The row cell to each variable offers a pull-down menu of options for summarizing the variable. For numeric variables the options are sum, average, count, maximum, minimum, standard deviation and variance. For text variables it is count.

The super table can be imported into Excel or another Access file. In fact, all modern software has standard functionality to import an Access super table as their source data. For Excel and Access, go to the Get Data icon and follow the guided trail. Of course, the super table can be developed further in all final destinations.

## **Super Tables from Operational Data**

All software, including “R,” provide the option to link to the super table in Access. Linking is often preferred over importing.

Imagine working with the table in Excel Pivot. We often discover that we want to modify the super table for a host of insightful reasons. We can pop the Access hood on the super table query and make the upgrades.

In turn, the background data to our Pivot will be refreshed upon command or the next time the Pivot file is opened. If we are doing a monthly update, the link will update the Pivot whenever it is opened for a session or refreshed during a session. Consequently, the analyst does not need to distribute the Pivot report upon every update.

### **4.4. Case 2: Seek Outlier Work Orders**

Plants are often limited to seeking outlier orders with crude methods. One is to investigate the top ten orders in cost or proxy to cost. The problem is that maybe some or ten are supposed to be the top ten. Exploring them reveals no opportunities.

With respect to the data of the super table, a better method is to seek the outliers with respect to cost center and maintenance type. Better yet is to include order lead craft type to the grouping.

Unfortunately, the example CMMS is not configured with a variable for lead craft. Craft lead is only sporadically noted in the description to each order. By the way they are noted in the description text, we could tease out the classifications with string functions. However, the classifications would only be partial because compliance has been less than 100 percent.

This section will demonstrate and explain aggregation variables. It will conduct a search for outliers to demonstrate aggregation in action. Two cases will be presented in this and the next section.

Case 2 of this section will demonstrate aggregation with the natural variables to the representative CMMS. Case 3 of the next section will apply aggregation to tease a craft lead variable out of the CMMS—find hidden variables. Although not demonstrated in the chapter, we would

## Chapter 4

want to join the classifications as a variable to the super table—making it more super.

However, let's note the alternative to aggregation in Access. The aggregations of Access are also available in Excel Pivot. Furthermore, Pivot is the first choice because it allows greater interactive and visual exploration than is natural to Access.

The difference is that aggregations in Pivot cannot deal with complex developments such as Cases 2 and 3. Of course, aggregations developed in the Access super table can be explored and further aggregated in Pivot.

### 4.4.1. Measurement for Outliers

We will apply a statistical test for outlier orders with respect to variance from average—Z-Score standardized. The idea is that we want to spot the orders with spending beyond some percent of all orders in a group of orders.

The Z-score test calculation is:

$$\text{Z-Score Standardized} = (R - \text{Avg}) / \text{Std Dev} \quad (\text{Eq 4-1})$$

**Where:**

**R** = Individual record.

**Avg** = Average or mean of the records to each group.

**StdDev** = Standard deviation of the records to each group.

For the demonstration of aggregation, we will assume the data is normally distributed. In life, we would test the assumption and act accordingly. Recall that Chapter 3 introduced and demonstrated graphic methods to test the assumption with histograms (Figure 3-7) and q-q plots (Figure 3-8). There is also the Shapiro-Wilk normality test function—`shapiro.test()`—in “R.” The set up and interpretation of the function can be found on the internet.

Next is the issue of deciding the percentage on which to base the Z-Score. It is a choice for the data-driven asset management organization to make. Table 4-8 provides scores associated with a range of choices.

The Z-scores will be associated with a group of orders. Case 2 is grouped upon cost center and maintenance type.

## Super Tables from Operational Data

**Table 4-8: Z-Score associated with percent outlier.**

Percent	Z-Score (+/-)	
	One-sided	Two-sided
90.0	1.29	1.65
95.0	1.65	1.96
99.9	3.10	3.27

The measured variable of interest to both outlier demonstrations will be craft hours. Hours, rather than dollars, are a better window into engaged maintenance capacity. Furthermore, hours submit well to related calculations such full time equivalent (FTE).

For the demonstrated cases, it will be assumed that the asset management operation wishes to investigate orders for which hours equal or exceed 95 percent of all orders to their respective groups. Accordingly, we will demonstrate with one-sided 95 percent. The associated Z-score is equal to or greater than 1.65.

If we were interested in orders beyond the two tails of the distribution, we would look for orders with a Z-score equal to or greater than 1.96 absolute. It is called the two-sided test.

We may exercise the two-sided test to seek oversized and undersized orders. Undersized orders may flag a self-inflicted fatal problem to ever achieving maximally effective and efficient asset management operations.

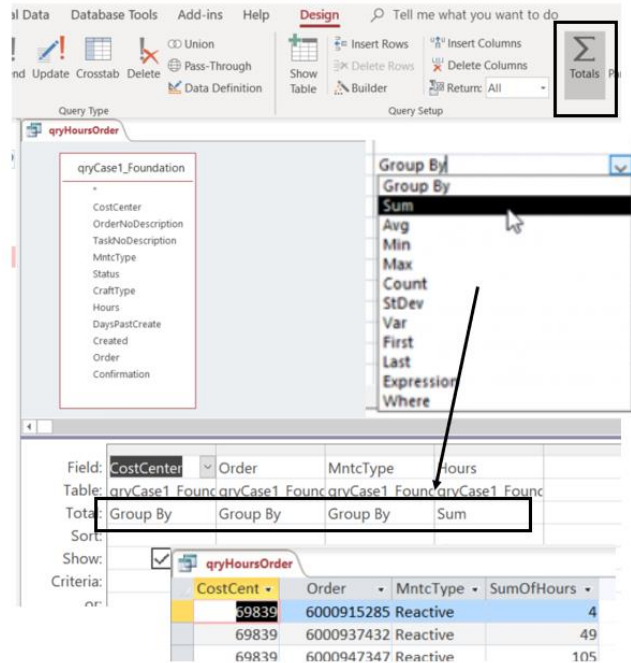
We may be observing the effects of craft hours not being accurately recorded to the orders that incurred them. This may, in turn, suggest that not all the orders at the upper extreme truly overran the work plan. Some have recorded to them hours engaged by other orders.

### 4.4.2. Activating Aggregation

Aggregation variables are built within a select query as shown in Figure 4-10. With a table or query in the design view, we check the Totals (summation symbol) icon in the Access ribbon. Figure 4-10 shows the action to develop the query qryHoursOrder and the resulting table. The code is provided by Table 4-9.

Upon doing so, notice the change in the design grid. The Total row appears.

## Chapter 4



**Figure 4-10: Clicking the summation icon adds a new row in the design grid.**

The lower portion of the figure shows the result. Without aggregation, each order would appear as a record for each individual craft engaged for each day of the order—time sheet records. Instead, there is a single record for each order. The hours incurred for all individuals to each order have been summed.

**Table 4-9: qryHoursOrder—join and grid code.**

<b>Join</b>	None: qryCase1_Foundation is sole source.			
<b>Field</b>	CostCenter	Order	MntcType	Hours
<b>Table</b>	qryCase1_Foundation	qryCase1_Foundation	qryCase1_Foundation	qryCase1_Foundation
<b>Total</b>	Group By	Group By	Group By	Sum
<b>Show</b>	Y	Y	Y	Y

In the Total row, for each variable, we select from ten options for summation—group-by, sum, average, min, max, count, standard deviation, variance, first and last.



## Super Tables from Operational Data

Notice in the figure and the code, we have pulled the foundation super table into the query. In the figure we have grouped on cost center, order and maintenance type.

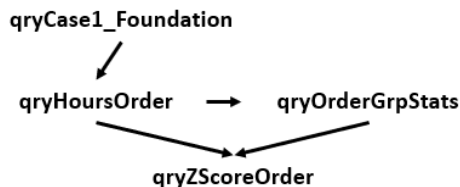
The Hours variable is aggregated with the option to Sum. However, we are not limited to one aggregation per variable. We can repeat the Hours variable as any one of the non-grouping options for aggregations. This will be seen as the demonstration unfolds.

Nor are we limited to a single non-group variable in the super table. For example, we could add another variable and select options for it (not demonstrated).

The possibilities of aggregation will appear in action throughout the remaining chapter. Most noteworthy are the immense ramifications. This is because so many insights involve calculations with group variables. The two cases of the chapter are only several of almost infinite possibilities.

### 4.4.3. Planning the Aggregation

The demonstrated foundation super table was a straightforward, follow-our-nose piece of work. However, sometimes it is good to flowchart what we are trying to do. Figure 4-11 is a flowchart of queries to be created and joined to arrive at the envisioned insight deliverable—qryZScoreOrder.



**Figure 4-11: Case 2 flowcharted to the final insight deliverable.**

From the super table, qryCase1\_Foundation, as its input, we will develop the qryHoursOrder object. It was previously shown in Figure 4-11 and followed with an explanation of its code.

## Chapter 4

Thence, we will develop a table of statistical elements to the Z-score computed with Equation 4-1—qryOrderGrpStats. The query will return the group averages and standard deviations.

Finally, qryHoursOrder and qryOrderGrpStats will be joined to generate qryZScoreOrder. The object will compute the Z-score for each order and return only those with a score equal or greater than 1.65.

### 4.4.4. Arriving at the Z-Score for Hours

The first aggregation query, qryHoursOrder, on the flow chart was previously introduced and explained. Next in the flow of things, we build qryOrderGrpStats. It will generate the group-level statistical elements to the ultimate Z-score calculation. As can be seen in Figure 4-12, qryHoursOrder is the source data to the statistical query.

Field:	CostCenter	MntcType	SumOfHours	SumOfHours	SumOfHours	SumOfHours
Table:	qryHoursOrder	qryHoursOrder	qryHoursOrder	qryHoursOrder	qryHoursOrder	qryHoursOrder
Total:	Group By	Group By	Count	Sum	Avg	StDev
Sort:						
Show:						
Criteria:	CostCent	MntcType	CountOfSum	SumOfSumOf	AvgOfSumOf	StDevOfSumOf
	69839	Prevent	19	224.5	11.82	9.55
	69839	Proactive	6	172.5	28.75	28.07
	69839	Reactive	10	255.5	25.55	31.47

**Figure 4-12: qryOrderGrpStats to generate stats for order groups of Cost-Center and MntcType.**

Table 4-10 provides the join and grid code to the qryOrderGrpStats object. No join is involved because the object is developed from a single input object—qryHoursOrder.

**Table 4-10: qryOrderGrpStats—join and grid code.**

Join	None: qryHoursOrder is sole source.		
Field	CostCenter	MntcType	SumOfHours
Table	qryHoursOrder	qryHoursOrder	qryHoursOrder
Total	Group By	Group By	Count

## Super Tables from Operational Data

Field	SumOfHours	SumOfHours	SumOfHours
Table	qryHoursOrder	qryHoursOrder	qryHoursOrder
Total	Sum	Avg	StDev

Once again, the groups are permutations of cost center and maintenance type. The summarizing variable is SumOfHours. Four aggregation variables have been created with it—count, sum, average and standard deviation. We could have given an alias to the lengthy automatically generated names. The pattern of the generated names are CountOfSumOfHours, SumOfSumOfHours, AvgOfSumOfHours and StDevOfSumOfHours.

The bottom portion of Figure 4-12 shows the returned table. Count, sum, average and standard deviation variables are returned. The full table is not shown, however, there are 11 permutations to the four cost centers and three maintenance types.

Recall that group average and standard deviation are elements to the Z-score calculation for each order. The third element are the hours to each order. Order is the record element to the Z-score calculation of Equation 4-1. Figure 4-13 shows the qryZScoreOrder object as the step to bring the three elements of the calculation together and return the Z-scores.

CostCent	Order	MntcType	OrderHrs	GroupCount	GroupHours	GroupAvg	GroupStdDev	ZScoreOrder
69839	6000974672	Prevent	36	19	224.5	11.82	9.55	2.53
69839	6000968570	Proactive	77	6	172.5	28.75	28.07	1.72
69839	6000947347	Reactive	105	10	255.5	25.55	31.47	2.52
69887	6000966188	Prevent	48	9	145.5	16.17	12.83	2.48
69887	6000966188	Prevent	48	9	562.5	28.13	40.29	3.40

Figure 4-13: Source, joins and returned table of qryZScoreOrder.

## Chapter 4

The joins and grid code to generate the table of Figure 4-13 is provided by Table 4-11. The new point to notice is how the three elements are brought together as a one-line calculation—Z-score.

**Table 4-11: qryZScoreOrder—joins and grid code.**

<b>Joins</b>	Inner: qryHoursOrder and qryOrderGrpStats on CostCenter Inner: qryHoursOrder and qryOrderGrpStats on MntcType		
<b>Field</b>	CostCenter	Order	MntcType
<b>Table</b>	qryHoursOrder	qryHoursOrder	qryHoursOrder
<b>Show</b>	Y	Y	Y
<b>Criteria</b>			
<b>Field</b>	OrderHrs: OrderHrs: SumOfHours	GroupCount: Coun- tOfSumOfHour	GroupHrs: Coun- tOfSumOfHour
<b>Table</b>	qryHoursOrder	qryOrderGrpStats	qryOrderGrpStats
<b>Show</b>	Y	Y	Y
<b>Criteria</b>			
<b>Field</b>	GroupAvg: Coun- tOfSumOfHour	GroupStdDev: CountOfSumOfHour	ZScoreOrder: ([SumOfHours]- [AvgOfSumOfHours])/[StDevOfSumOfHours]
<b>Table</b>	qryOrderGrpStats	qryOrderGrpStats	
<b>Show</b>	Y	Y	Y
<b>Criteria</b>			>=1.65

They are brought together by the double join shown Figure 4-13 and defined in Table 4-1. The alternative to the double join is to create an identifier in each table as a concatenation of cost center and maintenance type. Thence, we would join on the respective concatenated variables as the unique identifier.

There are reasons for the concatenation method. We may be getting a different outcome than expected from the double join. We may have stepped beyond what SQL can do. Another reason would be if we want to exercise other joins than the inner join.

To make the returned table clear, aliases have been given to otherwise the lengthy automatically generated variable names. Often it is best to set the aliases in the final step to the overall scheme.

## Super Tables from Operational Data

Finally, we arrive at the Z-score calculation. It is the calculated variable named ZScoreOrder. Because we seek the outliers to the groups among the 128 orders, we code a criterion for the Z-score variable. Although Figure 4-13 is only a partial view, the returned table reveals 11 outliers among the 11 groups. These are the orders an analyst would want to investigate rather than the top ten.

Once the queries are initially built for the first report month, the analyst can update each month's source CMMS tables and seek outliers. If the sourced CMMS tables are to be updated by append, the analyst may need to adjust one or more criteria to the queries—depends on the design. Once any required update tasks are done, any Pivot or other software linked to the queries will update (disseminate) when a new session is opened, or a refresh is called for in an open session.

The analyst may want to build the order group statistics upon longer periods such as the previous 12 months. The Z-score statistics—average and standard deviation—would be developed on history and then joined with the orders of the current report period.

### 4.4.5. Student's T-Score as Alternative

Because the primary purpose of the chapter is to demonstrate aggregation, we will stay with Z-score. However, this is a good place to introduce scoring with what is called the Student's T. If applied instead, the break point for being recognized as an outlier will be farther out from the average.

Just as for the Z-score, there is a one- and two-sided T-score associated with the chosen probability. The difference is that the score is influenced by the number of observations expressed through what is called degrees of freedom—observations minus one. Notice that the T-score converges on the Z-score as the count increases. We can expect that some of the previously flagged outliers will drop off the list for their respective groups.

**Table 4-12: Comparison of the Z- and T-scores for one-sided 95 percent.**

Group Count minus 1	Z-Score	T-Score
10	1.65	1.83

## Chapter 4

20	1.65	1.72
30	1.65	1.70
1,000	1.65	1.64

Because of the demonstration to this point, it is only necessary to describe the process to engage T-score rather than Z-score as the strategy to flag possible outliers. In fact, the paragraphs to follow will demonstrate to readers that they have become fluent in the language of super tables.

The calculation of the T-distribution is too complex to code as a calculated variable. The easy practical approach is to create a translation table. The translation table is then joined on the group count variable of `qryZScoreOrder`. We would then pull the T-score variable into the query. Any order with a Z-score equal or greater than the T-score would be investigated as an outlier.

Of course, there are many ways to develop a final filter for outliers. One is to calculate a variable of Z minus T and, in turn, filter upon records equal or greater than zero. This is also an example of a time when we may not want a variable to appear in the returned table. If so, turn off the Show option in the code grid.

Now for the translation table. It would be easy to build the table in Excel because it provides two functions for calculating the T-score value. They are `T.INV()` for one-sided and `T.INV.2T()` for two-sided. For both we enter group count minus one to the argument for degrees of freedom. For the probability of 95 percent as our example, we would enter 0.95 for `T.INV()` and 0.05 for `T.INV.2T()` as the respective probability arguments.

The variables to the translation table are count, one-sided for 95 percent, and two-sided for 95 percent. Other percentage categories can be added but for each there will be one- and two-sided variables. At each count, the argument for degrees of freedom will be the group count variable minus one.

The translation table can be located for anyone to link to or import from. Consequently, the count variable should reflect the greatest imaginable need for all users. Alternately, we could build an `IIF()` test in using queries to determine if the count is greater than 1,000 and use the Z-score

if true. Of course, we could easily build the translation table to range beyond 1,000 degrees of freedom and, thus, eliminate all complexity for its users.

### 4.4.6. Expression and Where Aggregations

Recall that Figure 4-10 revealed there are two options for aggregation which are obviously not aggregation. The rule of aggregation is that every field must have an aggregation. The Expression and Where options allow SQL, thus, Access to get around the rule.

The Expression option allows us to place a calculation in the fields row that is based on the aggregations in the design grid. The Where option allows us to filter on variables that are not in the design grid.

My observation is that rarely do the options arise in super tables. This is because there are so many ways to be direct such that the issues that may be dealt with at the level of aggregations are resolved upstream. See Alexander and Kusleika, page 316 for a deeper explanation and demonstration of the Expression and Where in aggregation.

## 4.5. Case 3: Seek Outliers with Lead Craft

Case 3 will extend the demonstration and explanation of Access skills by teasing a classification from the CMMS that it has not been configured to capture. We are seeking a variable that is hidden among others.

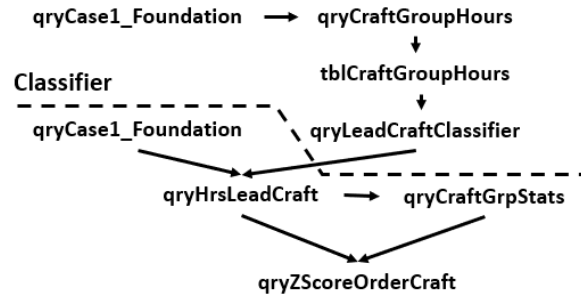
The ability to classify orders by lead craft is obviously fundamental to managing a maintenance operation. Without the classification, a maintenance SME would expect that averages calculated solely upon what is naturally provided by the representative CMMS are potentially misleading in the search for outliers.

A solution is an opportunity to demonstrate and explain the First-Last aggregations. We will look for outliers based on the groups of cost center, maintenance type and lead craft. Orders will be classified upon the assumption that the craft with the greatest hours to an order classify the lead craft.

## Chapter 4

### 4.5.1. Planning the Lead Craft Aggregation

As advised for Case 2, we should flowchart the sequence of queries to reach the end Z-score. Figure 4-14 flow charts the case. The next section will describe how each object is developed.



**Figure 4-14: Flowchart to Z-score for orders grouped on cost center, maintenance type and lead craft.**

Below the broken line, the sequence of queries that parallel Case 2. The difference is that the groups include lead craft. Above the line are additional queries that perform the lead craft classifications—the classifier.

### 4.5.2. Arriving at Z-Score with Craft Lead

The object `qryCraftGroupHours` is built with `qryCase1_Foundation` as its only input. Table 4-13 provides the join and grid code to build it.

**Table 4-13: `qryCraftGroupHours`—join and grid code.**

<b>Join</b>	None: <code>qryCase1_Foundation</code> is the sole source.		
<b>Field</b>	CostCenter	Order	MntcType
<b>Table</b>	<code>qryCase1_Foundation</code>	<code>qryCase1_Foundation</code>	<code>qryCase1_Foundation</code>
<b>Total</b>	Group By	Group By	Group By
<b>Sort</b>	Ascending	Ascending	
<b>Show</b>	Y	Y	Y
<b>Field</b>	CraftType	Hours	
<b>Table</b>	<code>qryCase1_Foundation</code>	<code>qryCase1_Foundation</code>	



## Super Tables from Operational Data

<b>Total</b>	Group By	Sum	
<b>Sort</b>		Descending	
<b>Show</b>	Y	Y	

The upper portion of Figure 4-15 shows what is returned by the code. Some orders have engaged multiple crafts. One such order is boxed in the figure. The hours for all engaged crafts have been summed by craft type. They are ordered from largest to smallest by virtue of the descend command for the Hours variable.

CostCenter	Order	MntcType	CraftType	SumOfHours
69839	6000915285	Reactive	Electrician	4
69839	6000937432	Reactive	Machinist	49
69839	6000947347	Reactive	Machinist	77
69839	6000947347	Reactive	Electrician	16
69839	6000947347	Reactive	Instr/Elec	9
69839	6000947347	Reactive	MultCraft	3
69839	6000957907	Prevent	MultCraft	4.5
69839	6000959888	Prevent	MultCraft	13.5
60920	6000961312	Prevent	MultCraft	7.5

CostCenter	Order	MntcType	FirstOfCraftT
69839	6000915285	Reactive	Electrician
69839	6000937432	Reactive	Machinist
69839	6000947347	Reactive	Machinist
69839	6000957907	Prevent	MultCraft
69839	6000959888	Prevent	MultCraft
69839	6000961312	Prevent	MultCraft
69839	6000961792	Prevent	MultCraft

**Figure 4-15: Hours sorted by craft hours and then selecting the first line to each order.**

The next step is to build the craft classifier query—qryLeadCraftClassifier. It is formed by running a query upon the returned data of qryCraftGroupHours.

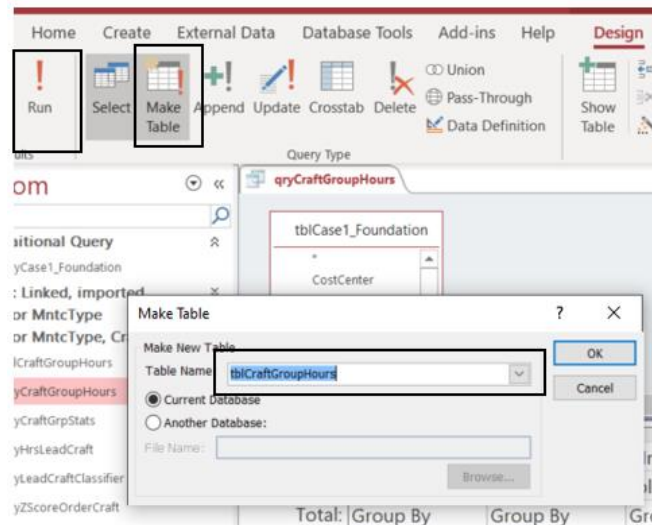
However, there is a problem if the qryLeadCraftClassifier object is built with qryCraftGroupHours as its source data. When we confirm the result in the returned table, we would find electrician shown as the lead craft even though we know it should be Machinist as shown in the upper portion of Figure 4-15.

There is something in our query that SQL cannot handle. Therefore, this is an opportunity to demonstrate a practical solution when things get strange.

## Chapter 4

Rather than spend time rebuilding and seeking the cause, there is a simple solution. Use the Make Table query. We will use it to generate a hard table from the qryCraftGroupHours and then use the hard table as the source data to the classifier—qryLeadCraftClassifier.

The process is to call up the query qryCraftGroupHours as shown in Figure 4-16. In the design view, select the Make Table icon on the ribbon. In the pop-up window give the hard table a name—tblCraftGroupHours (note the tbl prefix). Thence, click OK and then the Run icon and follow the steps to conclusion. As the final step, return the query to be a select query by clicking the Select icon; causing Make Table to turn off.



**Figure 4-16: The Make Table query in action.**

Like every aspect of Access, the reader can query the internet for a short YouTube video to watch the entire process in action. This several-minute task is advised for the first time through the process.

Notice in the flowchart that the hard table has been introduced between the source query and next query—qryLeadCraftClassifier. With tblCraftGroupHours as its sole source, the code is provided by Table 4-14.

## Super Tables from Operational Data

**Table 4-14: qryLeadCraftClassifier—join and grid code.**

<b>Join</b>	None: tblCraftGroupHours is the sole source.		
<b>Field</b>	CostCenter	Order	MntcType
<b>Table</b>	tblCraftGroupHours	tblCraftGroupHours	tblCraftGroupHours
<b>Total</b>	Group By	Group By	Group By
<b>Show</b>	Y	Y	Y
<b>Field</b>	CraftType		
<b>Table</b>	tblCraftGroupHours		
<b>Total</b>	First		
<b>Show</b>	Y		

Notice the aggregation option, First, in the Total row of the Craft-Type variable. Figure 4-14 shows what we seek. The machinist, as the greatest of craft hours in the upper portion of Figure 4-15 was the outcome of First as the aggregation. The first category in the categories of each order are returned. Consequently, the orders have been classified by lead type.

Let's pull the aggregations of Min and Max into the explanation. We could have used the Max aggregation in the classifier query. This would have eliminated the need for sorting in the qryCraftGroupHours object. The chapter leaves it to the readers to see if using the Max rather than First would have necessitated the hard table step.

If so, the Max option would be the preferable strategy. We should always strive to develop the trail of queries such that the final query is maximally automatic from data to final insight. For example, Case 2 is fully automatic, whereas, Case 3 requires the manual table step—a small act but not automatic.

Next is to build the craft lead to each order into the table of hours. This requires that qryCase1\_Foundation and qryLeadCraftClassifier be joined on the order variable. The join and grid code are detailed in Table 4-15.

**Table 4-15: qryHrsLeadCraft—join and grid code.**

<b>Join</b>	Inner: qryCase1_Foundation and qryLeadCraftClassifier on Order.		
<b>Field</b>	CostCenter	Order	MntcType
<b>Table</b>	qryCase1_Foundation	qryCase1_Foundation	qryCase1_Foundation

## Chapter 4

<b>Total</b>	Group By	Group By	Group By
<b>Show</b>	Y	Y	Y
<b>Field</b>	FirstOfCraftType	Hours	
<b>Table</b>	qryLeadCraftClassifier	qryCase1_Foundation	
<b>Total</b>	Group By	Sum	
<b>Show</b>	Y	Y	

Figure 4-17 shows the outcome. Notice that the work order that is boxed in the figure reports total hours (see Figure 4-15) for all crafts to the order. Meanwhile, machinist was identified as lead craft.

CostCent	Order	MntcType	FirstOfCraftT	SumOfHours
69839	6000915285	Reactive	Electrician	4
69839	6000937432	Reactive	Machinist	49
69839	6000947347	Reactive	Machinist	105
69839	6000957907	Prevent	MultCraft	4.5
69839	6000959888	Prevent	MultCraft	13.5
69839	6000961312	Prevent	MultCraft	7.5
60820	6000961707	Prevent	MultCraft	2.5

**Figure 4-17: Order is a machinist order with 105 hours for all crafts.**

Recall the Z-score calculation, Equation 4-1, requires the hours to each order. It also requires each group's average and standard deviation. Accordingly, the next step is to build a query—qryCraftGrpStats—to generate them. The code is provided by Table 4-16.

**Table 4-16: qryCraftGrpStats—join and grid code.**

<b>Join</b>	None: qryHrsLeadCraft is the sole source.		
<b>Field</b>	CostCenter	MntcType	FirstofCraftType
<b>Table</b>	qryHrsLeadCraft	qryHrsLeadCraft	qryHrsLeadCraft
<b>Total</b>	Group By	Group By	Group By
<b>Show</b>	Y	Y	Y
<b>Field</b>	SumOfHours	SumOfHours	SumOfHours
<b>Table</b>	qryHrsLeadCraft	qryHrsLeadCraft	qryHrsLeadCraft
<b>Total</b>	Sum	Count	Avg
<b>Show</b>	Y	Y	Y
<b>Field</b>	SumOfHours		

## Super Tables from Operational Data

<b>Table</b>	qryHrsLeadCraft	
<b>Total</b>	StDev	
<b>Show</b>	Y	

Figure 4-18 shows what is returned from the code. The statistics we need to compute the Z-score are the last two columns. Empty cells are the cases when there is only one order to a group.

CostCent	MntcType	FirstOfCraftT	SumOfSumOf	CountOfSumOf	AvgOfSumOf	StDevOfSumOf
69839	Prevent	Instrument	127	8	15.88	12.65
69839	Prevent	Machinist	16.5	1	16.50	
69839	Prevent	MultCraft	81	10	8.10	5.06
69839	Proactive	Electrician	17	2	8.50	2.12
69839	Proactive	Machinist	155.5	4	38.88	30.02
69839	Reactive	Electrician	23	3	7.67	6.35
69839	Reactive	Instrument	21	2	10.50	10.61

**Figure 4-18: Statistics for the 32 groups of cost center, maintenance type and craft.**

It is good practice to sanity-check returned tables. Here the sum of the hours and counts of orders can be confirmed as a test of the query. Click the Total icon in the records section of the ribbon. Then, at the bottom of the table window, confirm that the sum of hours and order counts match what we started with from the foundation super table.

As shown in the flowchart, now we will combine the hours of each of the 128 orders and the group statistics in a query that is developed to compute the Z-score. Furthermore, rather than look at all orders, the query will filter to orders equal or greater than the threshold Z-score. Table 4-17 provides the join and grid code to qryZScoreOrderCraft.

**Table 4-17: qryZScoreOrderCraft—join and grid code.**

<b>Join</b>	Inner: qryHrsLeadCraft and qryCraftGrpStats on CostCenter Inner: qryHrsLeadCraft and qryCraftGrpStats on MntcType Inner: qryHrsLeadCraft and qryCraftGrpStats on FirstOfCraftType.		
<b>Field</b>	CostCenter: Cost-Ctr3	Order	MntcType
<b>Table</b>	qryHrsLeadCraft	qryHrsLeadCraft	qryHrsLeadCraft
<b>Show</b>			
<b>Criteria</b>			
<b>Field</b>	Craft: FirstOfCraft-Type	OrderHrs: SumOf-Hours	GroupHrs: SumOfSumOfHours
<b>Table</b>	qryHrsLeadCraft	qryCraftGrpStats	qryCraftGrpStats

## Chapter 4

Show	Y	Y	Y
Criteria			
Field	GroupCount: CountOfSumOfHours	GroupAvg: AvgOfSumOfHours	GroupStdDev: StDevOfSumOfHours
Table	qryCraftGrpStats	qryCraftGrpStats	qryCraftGrpStats
Show	Y	Y	Y
Criteria			
Field	ZScore: ([SumOfHours]-[AvgOfSumOfHours])/[StDevOfSumOfHours]		
Table			
Show	Y		
Criteria	>=1.65		

What is returned by the joins and code are shown in Figure 4-19. The first thing to note is the triple join. As mentioned earlier, an alternative strategy is to concatenate the three join variables in both tables and then join on them.

CostCenter	Order	MntcType	Craft	OrderHours	GroupHours	GroupCount	GroupAvg	GroupStdDev	ZScoreCraft
69839	6000966841	Prevent	MultCraft	19.5	81	10	8.10	5.06	2.25
69887	6000946771	Reactive	Machinist	165	449	8	56.13	52.75	2.06
70107	6000966196	Prevent	Instrument	47	221.5	19	11.66	11.01	3.21
					240	7	34.29	36.91	2.11
					80	13	6.15	3.84	2.04

Figure 4-19: Z-score for lead craft and the triple join to create it.

## Super Tables from Operational Data

The Z-score calculation is placed in the field row. The threshold Z-score is placed as a criterion to the ZScore Field.

As presented for Case 2, we may choose to build a T-score variable. The chapter leaves it to the readers to extend what was described for Case 2.

Once again aliases are used heavily. They can be seen in the field cells to all variables. This makes the final product more readily consumed by those to which it is disseminated.

### 4.6. Comparison of Findings

As the final demonstration, let's compare the findings of the cases. We will compare approaches with respect to identifying outliers with ever sharper methods—top ten mostly costly, CMMS standard variables (Case 2) and with the added craft lead variable (Case 3).

The number of outliers is reduced from 11 to 5 when we seek outliers as Case 2 as compared to Case 3. Six of the outliers revealed by Case 2 also appear in the top ten. Two of the outliers revealed by Case 3 make an appearance. Cases 2 and 3 have three orders in common.

As mentioned earlier, having followed the chapter, its readers have become fluent in building super tables with Access. Therefore, this is the place at which the book will cut its readers loose. Rather than show the queries and codes to make the comparisons, the paragraphs to follow will describe how they can be done and leave the reader to execute.

The queries for Cases 2 and 3 have revealed a respective number of outliers. To determine how many appear in the top ten it is necessary to create a query to return the ten most costly orders.

With the Case1\_Foundation object as the source data to the query, create a sum aggregation on hours with orders as the group. While in the design view, find the Return box located in the Query Set Up section of the ribbon and enter ten.

Next, for each Z-score case, create a query to determine the number of orders that also appear in the top ten. The input data are the newly created top-ten query and the respective Z-score queries. Join on order.

## Chapter 4

Finally, we determine how many orders appear in both outlier groups. Pull the Z-score queries as source data into a newly created query. Make an inner join on order.

Of course, if we had extended the analysis to the T-score, there would have likely been different results. The reader is invited to extend Cases 2 and 3 to include the T-score and then make the same comparisons.

### 4.7. SQL in the Background

This is a good place to graphically and dramatically demonstrate why data-driven asset management has become a modern reality. Not too awfully long ago what the chapter has demonstrated required nothing short of a data scientist. Only they had the skills to write the necessary SQL code. Remember the days when we waited for IT to get our data?

We can see in the SQL code to develop the qryCase1\_Foundation object that specialized skills were a huge obstacle. Imagine being required to write the SQL code shown below; correct syntax and all. Or imagine the training that is required to be able to write the code.

```
SELECT tblWrkOrder.CostCtr3 AS CostCenter, [tblWrkOrder].[Order] & ": " & [OrdrShrtTxt] AS OrderNoDescription, [Task] & ": " & [TskShrtTxt] AS TaskNoDescription, tblMntcCode.MntcType, tblWrkOrder.Status, tblCraftInfo.CraftType, tblCraftHour.Hours, [DateFinish]-[Created] AS DaysPastCreate, tblWrkOrder.Created, tblWrkOrder.Order, tblWrkTask.Confirmation
FROM (((tblWrkOrder INNER JOIN tblWrkTask ON tblWrkOrder.Order = tblWrkTask.Order) INNER JOIN tblMntcCode ON tblWrkOrder.OrderType = tblMntcCode.[Order Type]) INNER JOIN tblCraftHour ON tblWrkTask.Confirmation = tblCraftHour.Confirmation) INNER JOIN tblCraftInfo ON tblCraftHour.PerNo4 = tblCraftInfo.EmplNum4
WHERE (((tblWrkOrder.CostCtr3)=69887 or (tblWrkOrder.CostCtr3)=70543 or (tblWrkOrder.CostCtr3)=70107 or (tblWrkOrder.CostCtr3)=69839) AND ((tblMntcCode.MntcType)="Prevent" or (tblMntcCode.MntcType)="proactive" or (tblMntcCode.MntcType)="reactive") AND ((tblWrkOrder.Status)
```



## Super Tables from Operational Data

```
Like ("*mcmp*")) AND ((tblWrkOrder.Created) Between  
#1/1/2011# And #2/27/2012#))  
ORDER BY tblWrkOrder.CostCtr3, [tblWrkOrder].[Order] & ": "  
& [OrdrShrtTxt], [Task] & ": " & [TskShrtTxt];
```

The modern reality is that an SME need not know a lick about the shown code. As an object is built in Access, its SQL code is automatically formed in the background.

Rather than locked in a sealed box, the code is still available to us. It can be viewed in the SQL option of the View icon. This is important because there is something to love about SQL code in the background.

What if for some reason we wanted to distribute the query but not as an Access file? We can by pasting the code in a txt file. The recipient only needs to create a new query, paste the code in the SQL view window and run the query.

Of course, Access is not the only software available to build super tables. The code also makes it possible to transfer what has been built in Access to other software, including “R.”. Because SQL is a universal language, the code can be pasted and run in any software’s functionality for SQL.

## 4.8. Administration of Tables

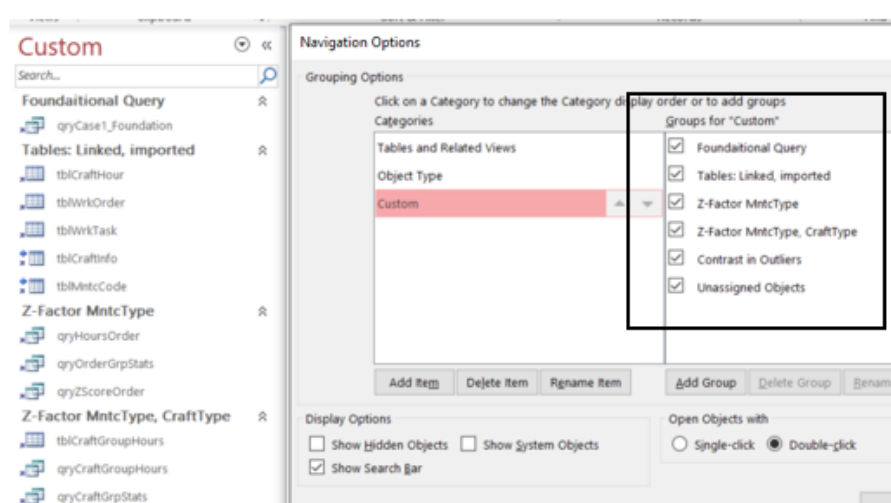
The section “Scenarios for Distribution” of the chapter presented seven steps to develop and manage super tables. The point of the seventh was that the query and hard tables rarely have importance as a one-off or to a single user. Accordingly, we should give thought to how they will be made available to everyone—done-by-one, used-by-many.

For example, the foundation super table serves almost any interest or as a source table to other super tables. Another example is the classifier query for lead craft. It is a table many would want to engage in their analyses.

However, let’s use this place to introduce administration within an Access file. The reason is accentuated by the left-most panel of Figure 4-20. There are 18 tables and queries involved in the chapter’s demonstrations and explanations.

## Chapter 4

Fortunately, we can organize them as shown in the right pane to the Navigation Options window. The window pops up upon right-clicking in the open space of the objects panel. Notice that the Access file has been organized to track the sequence of the chapter.



**Figure 4-20: Navigation capability to organize the objects of Access.**

In the window we have choices for organizing our objects—Tables and related views, Object type and Custom. They appear in the left-most panel.

By selecting Custom we can organize our tables and queries as we want. Below the panels of the pop-up window, categories can be added, deleted and renamed. The order of the categories can be established by dragging the titles.

Rather than dedicate a great deal of book space, the readers are advised to query the internet for a YouTube demonstration. Search the internet for “access navigation panel YouTube.”

## Bibliography

Alexander, Michael and Kusleika, Richard. Access 2016 Bible. John Wiley & Son, Inc. 2016. Parts 3 and 4.

## **Super Tables from Operational Data**

Oesko, Suljan. Pivot Tables In-Depth for MS Excel 2016. Suljan Oesko. 2017.