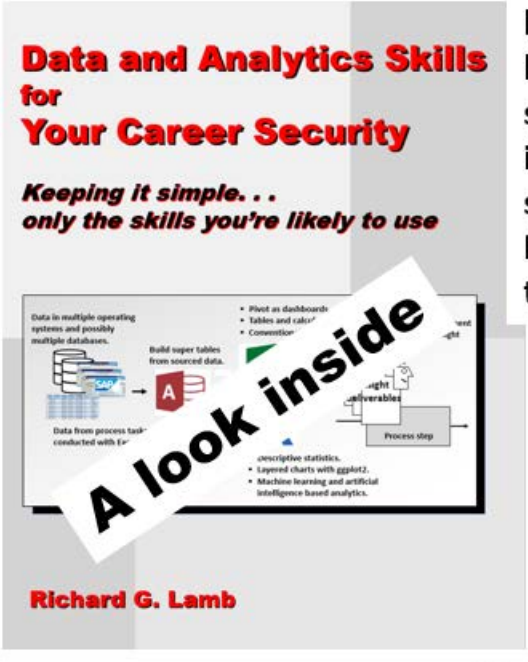

Your career security will leap forward when you learn to work with MS Access to build what the book calls super tables. Access makes building super tables easy with the intuitive tabular interface to SQL called Query by Example. The skills learned transfer to any software, e.g., Power BI. And all software have the functionality to import your super tables directly from Access.

Excerpts Chapter 3:

3.3. Case 1: Foundation Super Table

3.4. Case 2: Seek Outlier Work Orders (partial)

Additional "Look Inside" at https://analytics4strategy.com/book-look-inside
Book is available from Amazon
Paperback, 508 pages, 300 figures and outputs, 7.5 x 1.15 x 9.25 inches, 2.37 pounds

Chapter 3 Super Tables from Operational Data 43
3.1. Perspective 44
3.2. Scenarios for Demonstration 45
3.3. Case 1: Foundation Super Table 47
3.3.1. Identify, Extract and Import or Link 47
3.3.2. Translation Tables 49
3.3.3. Overview of the Query Process 50
3.3.4. Joins and Their Ramifications 53
3.3.5. Coding the Design Grid 56
3.3.6. Generate the Super Table 61
3.4. Case 2: Seek Outlier Work Orders 63
3.4.1. Measurement for Outliers 63
3.4.2. Activating Aggregation 65
3.4.3. Planning the Aggregation 66
3.4.4. Arriving at the Z-Score for Hours 67
3.4.5. Student's T-Score as Alternative 70
3.4.6. Expression and Where Aggregations 71
3.5. Case 3: Seek Outliers with Lead Craft 72
3.5.1. Planning the Lead Craft Aggregation 72
3.5.2. Arriving at Z-Score with Craft Lead 72
3.6. Comparison of Findings 78
3.7. SQL in the Background 79
3.8. Administration of Tables 80
Bibliography 81

3.3. 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.

3.3.1. 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 3-1 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.

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.

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.

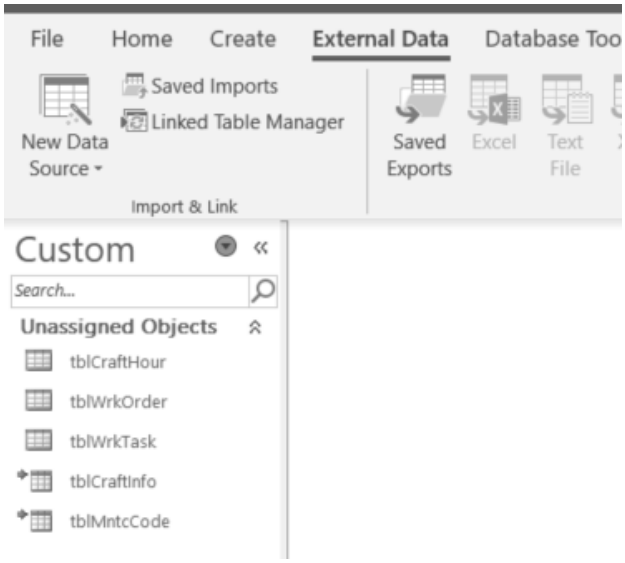

Figure 3-1: Three downloaded tables and two clarifying tables.

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 3-2, the Select query will open automatically with the Show Table pop-up window.

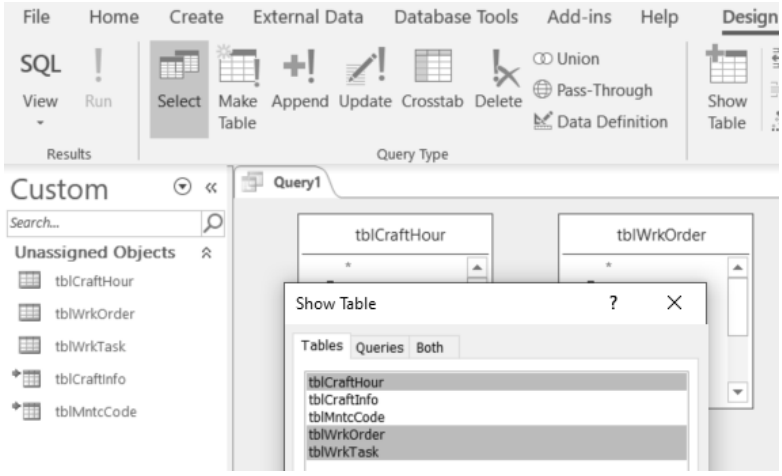

Figure 3-2: Tables imported and linked to Access for building super tables.

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.

3.3.2. 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.

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

tblMntcCode

ID	Order Type	MntcType	MntcDesciption
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

Figure 3-3: 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.

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.

3.3.3. 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 3-4) 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.
- **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 3-4 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.

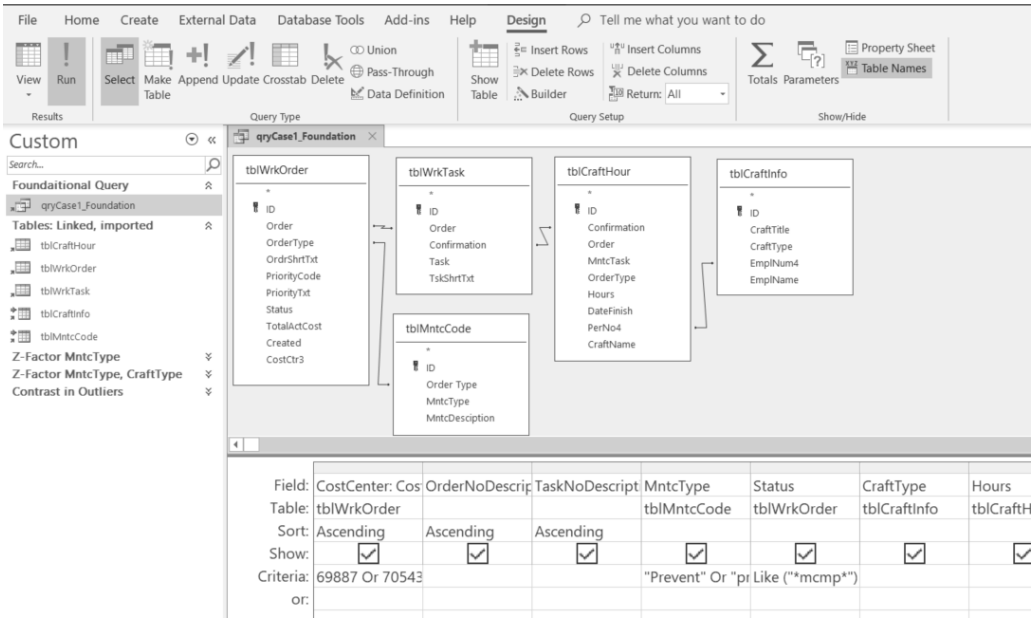

Figure 3-4: Design view to the super table.

Figure 3-5 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. Section 3.8 will explain the objects view and how objects are organized as shown in the figure.

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.

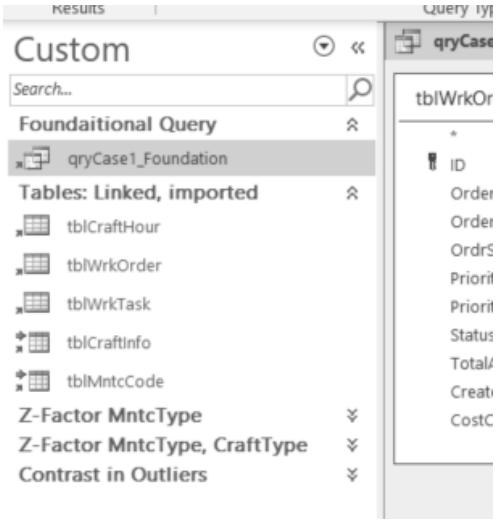

Figure 3-5: List of objects associated with the super table.

Now let's look at how the sub tables are joined in the super table. Figure 3-6 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.

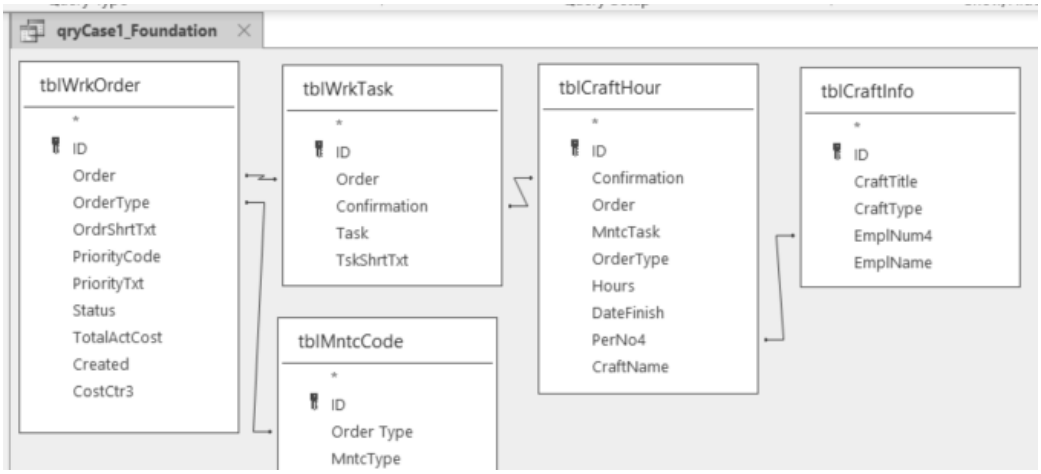

Figure 3-6: All five tables to the super table and their joins.

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.

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 3-4 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 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.

3.3.4. Joins and Their Ramifications

Figure 3-6 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 will use Figure 3-7 to explain them. Then we will see how to make the choices for our super table.

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.

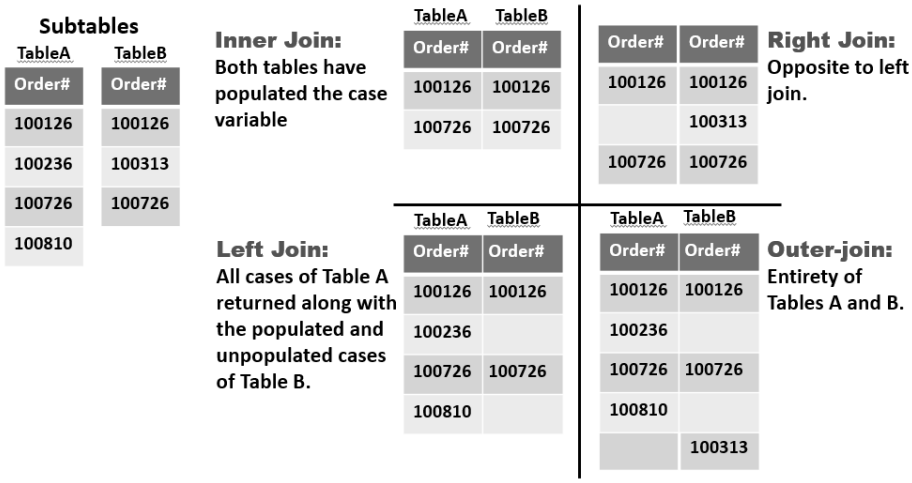

Figure 3-7: Tables can be joined to return four different outcomes.

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.

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 3-8 shows how we set the joins between tables with Access. The process begins with a right-click on a join line in Figure 3-6 and selecting the Join Properties option. The Join Properties window shown in Figure 3-8 will pop up for specifying the join we want.

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.

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. Chapter 11 will demonstrate the power of outer joins to analyze work attainment.

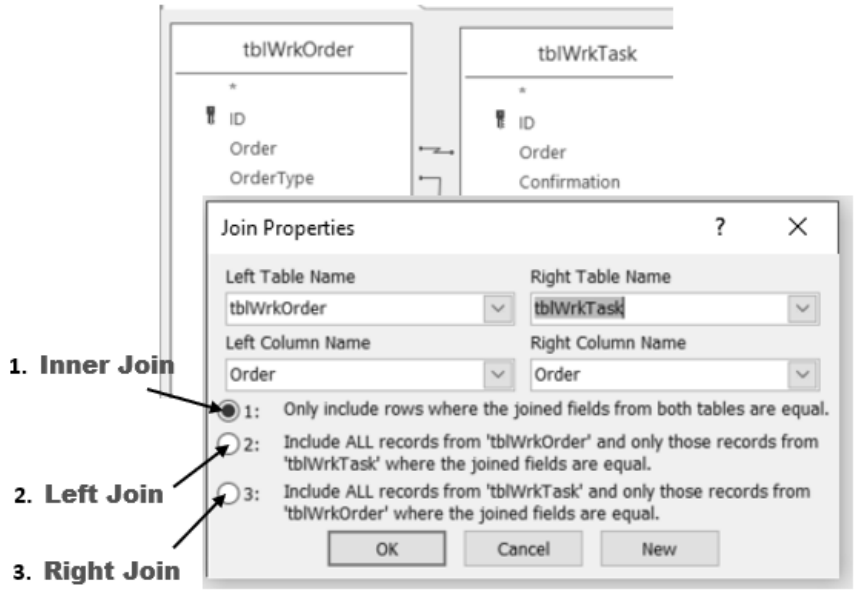

Figure 3-8: Select the join type, if not the default inner join.

The joins to the super table, qryCase1_Foundation, are provided in Code 3-1. They are presented as they would appear in the Join Properties pop-up window (Figure 3-8).

Code 3-1: 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 Excel, may eventually grow to hold more records than Excel has the capacity to hold.

3.3.5. 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 Code 3-2. 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.

Code 3-2: qryCase1_Foundation—grid code.

Join	See Code 3-1		
Field	CostCenter: CostCtr3	OrderNoDescript ion: [tblWrkOrder].[Order]&": "&[OrdrShrtTxt]	TaskNoDescription: [Task]&": "&[TskShrtTxt]
Table	tblWrkOrder		
Sort	Ascending	Ascending	Ascending
Show	Y	Y	Y
Criteria	69887 Or 70543 Or 70107 Or 69839		
Or			

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

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 Code 3-2) 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.

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.

Figure 3-9 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.

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.

Figure 3-9: Criteria expressions for all data types.

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 3-4 and Code 3-2 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 Code 3-2—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 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 3-4 and Code 3-2—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 Code 3-2.

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

Figure 3-10 provides the list of available text criteria. We all know of many or all of them from our experience with Excel and other software.

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

Figure 3-10: Criteria for working with text strings.

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 using the first of them. They are the conditional expressions—IIF() and Switch(). They are coded as shown in Figure 3-11.

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

Figure 3-11: The conditional expressions IIF() and Switch().

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. Figure 3-12: shows them for date variables and Figure 3-13 for numeric 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

Figure 3-12: Criteria for working with date 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

Figure 3-13: Criteria for bounding numeric variables.

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.

3.3.6. 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 3-4. It returns the super table as shown in Output 3-1. 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.

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.

qryCase1_Foundation

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

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

Output 3-1: The table returned from the code.

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.

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.

3.4. 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 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 the 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.

3.4.1. 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:

Z-Score Standardized = (R - Avg)/Std Dev (3-1)

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. Chapter 4 introduces and demonstrates graphic methods to test the assumption with histograms (Figure 4-7) and q-q plots (Figure 4-8). 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. Figure 3-14 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.

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

Figure 3-14: Z-Score associated with percent outlier.

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.

3.4.2. Activating Aggregation

Aggregation variables are built within a select query as shown in Figure 3-15. With a table or query in the design view, we check the Totals (summation symbol) icon in the Access ribbon. Figure 3-15 shows the action to develop the query qryHoursOrder and the resulting table. The code is provided by Code 3-3, qryHoursOrder.

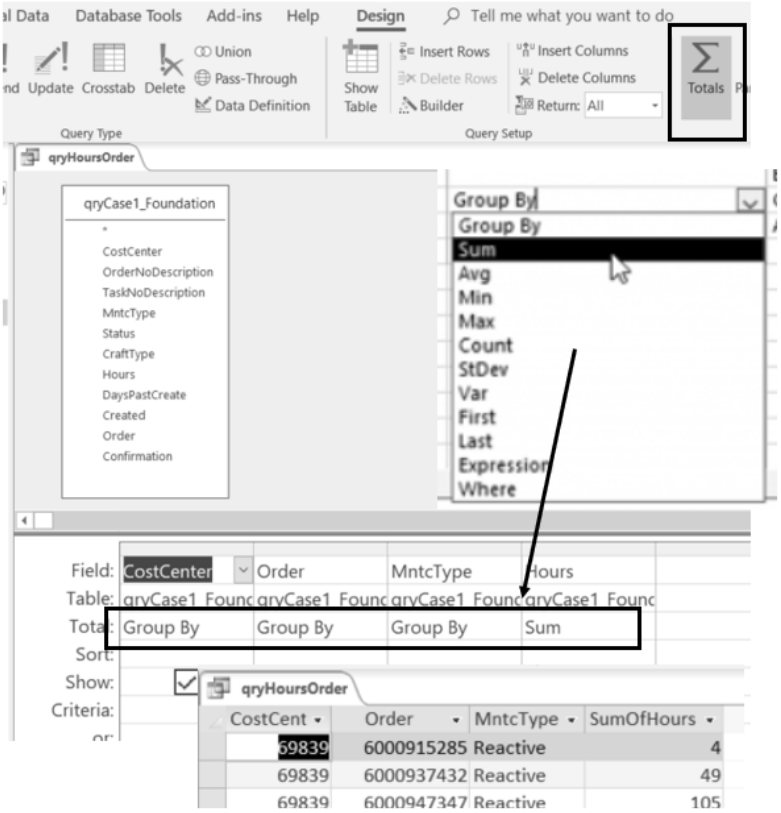

Figure 3-15: Clicking the summation icon adds a new row in the design grid.

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

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.

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.

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.

Code 3-3: qryHoursOrder—join and grid code.

Join	None: qryCase1_Foundation is sole source.			
Field	CostCenter	Order	MntcType	Hours
Table	qryCase1_Foudation	qryCase1_Foudation	qryCase1_Foudation	qryCase1_Foudation
Total	Group By	Group By	Group By	Sum
Show	Y	Y	Y	Y

3.4.3. 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 3-16 is a flowchart of queries to be created and joined to arrive at the envisioned insight deliverable—qryZScoreOrder.

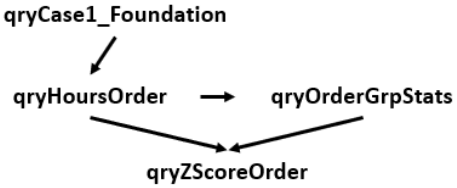

Figure 3-16: Case 2 flowcharted to the final insight deliverable.

From the super table, qryCase1_Foundation, as its input, we will develop the qryHoursOrder object. It was previously shown in Figure 3-16 and followed with an explanation of its code.

Thence, we will develop a table of statistical elements to the Z-score computed with Equation 3-1—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.

Data and Analytics/Operational Effectiveness

Data and Analytics Skills for Your Career Security

Keeping it simple, only the skills you're likely to need

Richard G. Lamb

For those of us who are role holders in enterprise functioning, the personal purpose of acquiring practical working skills in data and analytics is to be able to better do what we already do and find new ways to do better yet. It follows that if you are a role holder who brings and incorporates data and analytics methods in your thoughts and tasks, your career outlook will be more secure and exciting. The book is written to be your gateway to the skills and to be the templates with which you will install the methods in your operational roles.

We all know that the field of data and analytics is huge and intimidating. It is a long slog to becoming comfortable. During the author's own long slog until arriving at the book, something exciting bubbled to the surface. There is a big difference between what we need to know and everything there is to know. We need to know what is possible as insight for decisions and functioning, we need to know how to get to the insight and, finally, we need to be able to interpret the insight. Just as the book does, we can leave the rest to the data scientists.

About the Author: In 2003, Richard Lamb, while struggling to get at the history captured in the databases of operational systems, found the skills to extract datasets of related history and join them in a super table of variables to make possible what was being envisioned for operational effectiveness. In 2014, Richard realized that, with statistical analytics and free enabling powerful pc-level software, an enterprise could ask and answer questions of operational effectiveness that are otherwise not possible. His activism to bring the epiphanies into the careers of role holders in the mainstream of operations has arrived at this book to explain data and analytics through the demonstration of methods.

Richard is a Registered Professional Engineer and Certified Public Accountant. He has previously authored two books: Availability Engineering and Management for Manufacturing Plant Performance, and Maintenance Reinvented for Business Performance. He has a BSCE, BBA and MBA from the University of Houston and a graduate-level Applied Statistics Certificate from the Texas A&M University.

https://analytics4strategy.com/data-and-code

ISBN 978-1-7343947-0-2

90000>

9 781734 394702

Analytics4Strategy