Using Tabular Data in Snap Transcript

Tables are the best way to store the complex information that drives your configurator.

While tabular data can be stored in different places, moving data in and out of them is the same: you use queries to read and write tabular information.

Gain a basic understanding of how queries work. See how a simple query to read data is built, and how to use the result set from that query.

Objectives

- Tables and Queries
- Creating a Query
- Using Results

Tables and Queries

Data is often stored in either a hierarchical structure called a tree, or a grid-like structure called a table. Let's discuss tables, and how to get information into and out of them.

In Epicor CPQ, there are 3 of these grid-like table structures. They are local tables, database tables, and the results of your own functions, called type arrays. If you're not already familiar with these basic concepts, review the course introducing Local Tables. It's important to know which table structure is best for your data's size, use, and security needs.

Regardless of what table structure is used, Snap logic can put information into these tables, or read data back out. We call this process "querying" a table, because it's most common to ask, or "query" a table to read data from it.

One way of querying data and placing it directly into the options in your user interface is called an Option Filter. Review the course introducing Option Filters if you're not already familiar with this technique. It's a powerful technique that requires little effort.

But what if you want to gather information from a table for other purposes? For example, maybe you want a validation rule to look up the tolerances for the current design, to see if the user's selection is within those tolerances? Or if you want a pricing rule to look up the cost of a material used in the current design?

These are examples best answered with a second way of querying data. Not querying data through an option filter, but through Snap code.

The Snap code applies your query to the source table, and receives back a result. This result is usually the rows that match your query. It could be one row… many rows… or no rows at all that match. Regardless of the size, the result is the answer we get back. But that result can be one of three different things.

If we asked for any rows that match, the result is an array. Arrays can store any number of things, and give you ways to loop through all of them. See the documentation on arrays for more on how to manage them.

Sometimes you may write a query that asks for just a specific item, and limits the number of results to just one. The result to this sort of a query is not an array. It's just the one row from the table that matches.

Finally, queries can fail. You may ask for something that doesn't exist. In this case the result is a special object, called "null". Null means nothing. It is a clear way to know that no information in the source matches our query.

Since reading data is the most common query, that's the focus of this course. You should know Snap can not just read, but also update existing rows in a table, create new rows, or delete rows. For more information on these other query operations, look at the online documentation.

Regardless of which type of table you want to query, the Snap language is the same. For this course, we'll use a local table as our source. But you can easily change that source as you need.

Select the right arrow icon to move to the next topic.

Creating a Query

Let's write our first query. We want to see if the current configuration has been ordered in the past. If so, and it already has a part number, then we would like to re-use that part number. There's no need to create a new one.

When do we want this lookup to happen?

There's no need to calculate the part number with every click the user makes. This business logic would be best run in a Submit rule. But for easier testing, let's build the logic in a Value rule. Once it works, it's easy to copy and paste into a Submit rule later.

What is the table we will be querying?

Our product manager gave us this data, which we've placed into a local table for our first prototype. Later, we may move this into a database table for both reading and writing.

Does the table have the information we need?

Cubes have 4 unique parameters: height, width, depth, and material. This table has all four, and can be used for our lookup. If we find a matching row, we want to return the part number back. And if there is no matching row, we know we will get "null" as our result.

First, be sure to have a text field in our configurator UI called "f-PartNumber".

Create a new value rule, with the name of "get Part Number".

Drag the "query" block in the toolbox onto the workspace. It starts with the word "declare".

The query block gives us a result (a yellow variable), and a Type Name (a white type). The type is not often used, and is described in another course on Types. We are interested in the result. To make our code easier to understand, change the name from "result" to "PartNumberResult".

That's what comes out of the query block. But what goes into it? We need to specify the source of data. Click once on the query block to see the buddy blocks appear on the right. Drag in the Table block, and pick the source table.

As it stands now, this query will give us all the rows from the source table. We need to filter it. Click the mutation symbol on the left side of the query block, and add a "where" slot.

The syntax for queries is the same as what you've already learned for option filters. Use the "and" logic block, the "column compare" block, and the "get field block" to add your logic. For more information on how to write a basic query, review the courses on option filters.

As it stands now, this query will just give us the matching rows. What if there's a problem in the source table, and two rows both claim to be the part number for a configuration? We don't want both of them.

Apply a limit to the query. Add a "select" slot, and choose "first". Now we are guaranteed to get a maximum of one rows returned, even if there are multiple matching rows.

Currently, we haven't specified in the "select" slot which columns of data we want returned from the table. By default, the result will have all columns. Often, this is just fine. If we wanted to only get specific columns back in our result, we could specify them here.

For our query, we'll include all the columns.

Finally, we said we want just the first row, if there are multiple matches. But what defines "first"?

Use the mutation symbol to add an "order by" slot. Order the results in ascending order, by the date they were first configured. In this way, we will honor the oldest part number, and ignore any more recent duplicates that may have been added in error.

Our query is complete. This will give us any row from the table that matches our customer's cube parameters. If there's no match, PartNumberResult will be null. If there's more than one match, we know we'll just get the oldest, best one.

Other mutations can be added to the query block: you can learn more about them in the online documentation. Snap uses the same SQL language used by just about every database system.

Select the right arrow icon to see how we can use the query results.

Using Results

You've written your query. The result is ready. But how do we use the result of a query?

The result is yellow, which means it's a variable. You already know that variables have a short lifetime (they exist only until the final rule of this rule type is run), and the "get" block helps us use them.

We can simply get the result, and write it into our part number field with a setField block. Notice that the result is a complex object: it isn't just text or number. It's a collection of the columns we selected. So, pick the right column you want to place into the part number field.

We're almost done. But whenever dealing with query results, keep in mind that they are special: you don't know if your result is data, or if it's nothing at all because no rows matched.

We need to test our result first, before we place it into this field.

Use a logic block to see if the result is null. If so, there's nothing to write into the part number field. So we'll have to get our part number from somewhere else, or calculate it ourselves. Here, we'll just hard-code it. We can use other logic later on to fill that in.

Our work is done. Now, as the user changes any parameter, the part number will change. Test and confirm your work.

Finally, you may want to use the "copy JSON" feature to copy this snap code out of a value rule, and into a Submit rule where it belongs.

If your query result returns many rows, instead of just one like we do here, use array functions to loop through the result. See the online documentation for more info.

And if the source of your data changes from a local table to a database table or other structure in the future, just replace one Snap block. No other changes are required.

Recap

You've seen how all of the different table structures can be queried using one standard Snap block: the "query" block.

You've written a query, selecting just the rows and columns you want, with filters that limit the number of rows and sort them into the correct order.

And you've seen how you can use the result of the query to fill in a field or perform other work, keeping in mind that query results can be something or nothing at all.

About Epicor

Epicor Software Corporation drives business growth. We provide flexible, industry-specific software that is designed around the needs of our manufacturing, distribution, retail, and service industry customers. More than 40 years of experience with our customers' unique business processes and operational requirements is built into every solution—in the cloud, hosted, or on premises. With a deep understanding of your industry, Epicor solutions spur growth while managing complexity. The result is powerful solutions that free your resources so you can grow your business. For more information, connect with Epicor or visit www.epicor.com.

EPICOR

Corporate Office
804 Las Cimas Parkway
Austin, TX 78746
USA
Toll Free: +1.888.448.2636
Direct: +1.512.328.2300
Fax: +1.512.278.5590

Latin America and Caribbean
Blvd. Antonio L. Rodriguez #1882 Int. 104
Plaza Central, Col. Santa Maria
Monterrey, Nuevo Leon, CP 64650
Mexico
Phone: +52.81.1551.7100
Fax: +52.81.1551.7117

Europe, Middle East and Africa
No. 1 The Arena
Downshire Way
Bracknell, Berkshire RG12 1PU
United Kingdom
Phone: +44.1344.468468
Fax: +44.1344.468010

Asia
238A Thomson Road #23-06
Novena Square Tower A
Singapore 307684
Singapore
Phone: +65.6333.8121
Fax: +65.6333.8131

Australia and New Zealand
Suite 2 Level 8,
100 Pacific Highway
North Sydney, NSW 2060
Australia
Phone: +61.2.9927.6200
Fax: +61.2.9927.6298