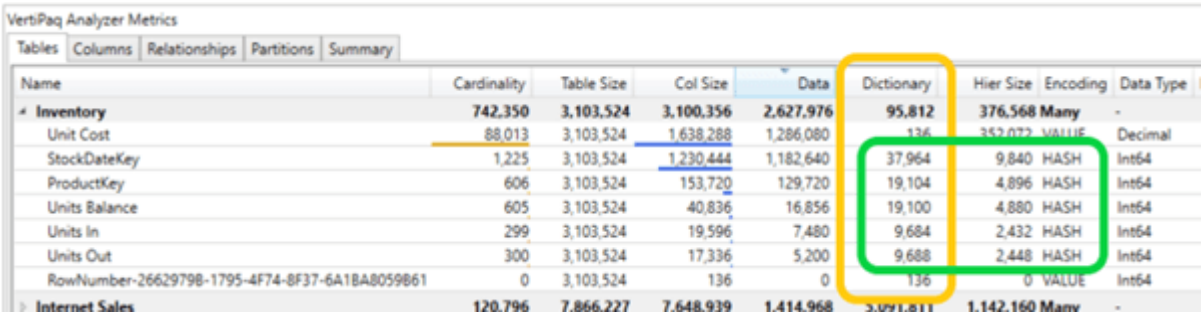
Manage your fragging VertiPaq dictionary!

With organisations starting to realise the many benefits of Power BI Premium Capacities and utilising the golden dataset approach, more and more we're pushing up against dataset size limits. Here's a nice little-known trick thatw will often free up valuable gigabytes in your dataset.

First, a little background

You're probably already aware of the VertiPaq Analyser – if not, stop what you are doing right now and go check this out! VertiPaq Analyser makes it easy to see what's in your dataset and how much space it takes. Dax Studio has a nice version of this built in, with is my personal go-to.



Name	Cardinality	Table Size	Col Size	Data	Dictionary	Hier Size	Encoding	Data Type
Inventory	**742,350**	**3,103,524**	**3,100,356**	**2,627,976**	**95,812**	**376,568**	**Many**	-
Unit Cost	88,013	3,103,524	1,638,288	1,286,080	136	352,072	VALUE	Decimal
StockDateKey	1,225	3,103,524	1,230,444	1,182,640	37,964	9,840	HASH	Int64
ProductKey	606	3,103,524	153,720	129,720	19,104	4,896	HASH	Int64
Units Balance	605	3,103,524	40,836	16,856	19,100	4,880	HASH	Int64
Units In	299	3,103,524	19,596	7,480	9,684	2,432	HASH	Int64
Units Out	300	3,103,524	17,336	5,200	9,688	2,448	HASH	Int64
RowNumber-26629798-1795-4F74-8F37-6A1BA8059B61	0	3,103,524	136	0	136	0	VALUE	Int64
Internet Sales	**120,796**	**7,866,227**	**7,648,939**	**1,414,968**	**3,091,811**	**1,142,160**	**Many**	-

You'll notice in VertiPaq output there is a "Dictionary" column – you may also see that it tends to be larger when we have an encoding of type "HASH" on the column.

HASH encoding is one of the techniques VertiPaq uses to compress data, limiting the amount of valuable space consumed values that repeat in a column. Instead of directly storing the values, an index of all the unique values for that column are stored and that index is what actually gets used in the table.

These column value indexes are maintained in the dataset's dictionary. It's like a hidden extra layer of columnar normalisation in the model.

E.g. If this is how a column of data looks when we query our table:

My Column
123A
XYZ321
XYZ321
123A

Here's what's actually going on in the table storage and dictionary with HASH encoding:

Table: My Column
1
2
2
1

Dictionary: HASH Index	Dictionary: My Column
1	123A
2	XYZ321

The majority of columns in most dataset use this kind of encoding. There are other encoding types, such as Value Encoding and Run-Length encoding, but these don't add significantly to the dictionary size like Hash encoding does.

Dictionary bloat

Now, sometimes as you are incrementing the data in your fact tables or loading and unloading data in partitions etc, you'll see that dictionary size starts getting a little too big. This can happen because the memory required to perform the dictionary compression is allocated in advance of the refresh. Unused dictionary memory may not be released at the end of the process. Fact tables with incremental refresh are particularly prone to this type of bloat.

This will impact the overall size of the dataset, causing it to take up more memory than needed. The problem is exacerbated if your refresh process not only adds new rows but also removes older rows. In extreme cases this may amount to a sizable bloat, perhaps the difference between on-demand refreshes working and failing due to space constraints.

But fear not, there's a way to recover this dictionary space:

Defragment refreshes

To ensure the dataset and its dictionary are reduced to optimal size and to make sure the VertiPaq engine can efficiently fetch data, it is advised to periodically defragment the tables.

This is similar to how we used to need to defragment our computer hard drives (these days this is done automatically), essentially rebuilding the file index and moving data around to create as much contiguous file space as possible.

In this case we're dropping and rebuilding the dictionary, re-indexing and reclaiming space that had been allocated but unused. The dataset defragmentation process does not re-load any data from the source, rather the dictionary is rebuilt from the data already saved in the dataset. The reclaimed space can be significant. I have personally witnessed a dataset drop from 12.8Gb to 6.4Gb with a simple defrag.

How it's done

Using XMLA endpoints, we can connect to a Power BI dataset and run commands from SQL Server Management Studio. If you are unfamiliar with Tabular Model Scripting Language (TMSL), see this Microsoft learn page on how to script dataset refreshes.

A simple example of a script to defragment a single table is below:

```
{
  "refresh": {
    "type": "defragment",
    "objects": [
      {
        "database": "Enterprise Golden Dataset",
        "table": "Big-Ass Table"
      }
    ]
  }
}
```

The script can be modified to accommodate a full list of all tables, effectively defragmenting the entire model.

Where to next?

If you find you need to regularly defrag your dataset, this could be an indication that there is something larger that needs to be addressed in how you are incrementing your data. Are you, for example, regularly updating large text fields, or doing daily surrogate re-indexing with GUIDs? These things will contribute to dictionary bloat and may be a sign that a new approach should be considered.

Are you having trouble taming and optimising your large Power BI dataset models? We can help. We specialise in Data and can help you with designing and implementing Data Models to suit your project. Please contact a Fujitsu Data & AI specialist now.

Contact

Fujitsu Data & AI

+61 3 9924 3000