Disclaimer: I work for Microsoft. More specifically, I work for Microsoft Advertising. We sell the ads that are served on Microsoft websites. Mostly Bing. Much to my chagrin, I don’t have any of that [sweet sweet inside information](https://twitter.com/GuyInACube/status/1073692665571655681) about Power BI. This blog represents my own view and not Microsoft’s.

# Part 1: Manual Aggregations in DAX

Aggregations are one of my favorite new features added to Power BI in 2018. [Adam thinks so too](https://twitter.com/GuyInACube/status/1073693688155529216). The quick summary is that if you have two tables about the same facts at different levels of detail, Power BI can intelligently choose which one to use for each query to get the best performance. This article is about aggregations. If you don’t already know about them, you should read more about them [here](https://docs.microsoft.com/en-us/power-bi/desktop-aggregations) and [here](http://radacad.com/power-bi-fast-and-furious-with-aggregations) and lots of other places.

This article isn’t about using Aggregations. It’s about how you can…not use them. I’m going to explain how to create “aggregations” without using the Aggregations feature.

Using Dax.

 Of course.

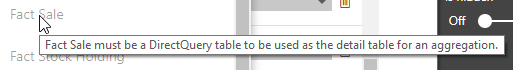
I’m also going to tell you why you might want to do that. It might get a little confusing, so I’m going to call these new measures Manual Aggregations. In contrast with UI Aggregations.

## Why?

I can think of four reasons.

First, no plans have been announced for UI Aggregations to be available in Analysis Services.

Second, today you cannot set up an [aggregation over a table in Import storage mode](https://docs.microsoft.com/en-us/power-bi/desktop-aggregations#validations).

[](https://1.bp.blogspot.com/--jH9TMvyp_M/XChy0Xf6CtI/AAAAAAAAeC0/w_VwoZ785dIdPnm95imMm0oRKYsuq0W2ACLcBGAs/s1600/DQ%2BRequired.png)

Third, today you cannot use [row level security with Aggregations](https://docs.microsoft.com/en-us/power-bi/desktop-aggregations#validations).

Fourth, doing things you probably shouldn’t using [DAX is fun](http://radacad.com/author/phil).

Reasons two and three will be addressed by the time the feature goes GA. And there are also no plans I've seen for Analysis Services to support Composite models, which really make aggregations shine.

RLS is how I came upon this topic. I needed to secure different granularities differently. Which is a great capability and has nothing to do with something as amorphous as performance.

There are many different ways you might want to control granularity security so I expect this technique to still be valuable even when RLS and UI Aggs work together. Which is why I'm writing about it. Also because doing things you probably shouldn't using DAX is fun.

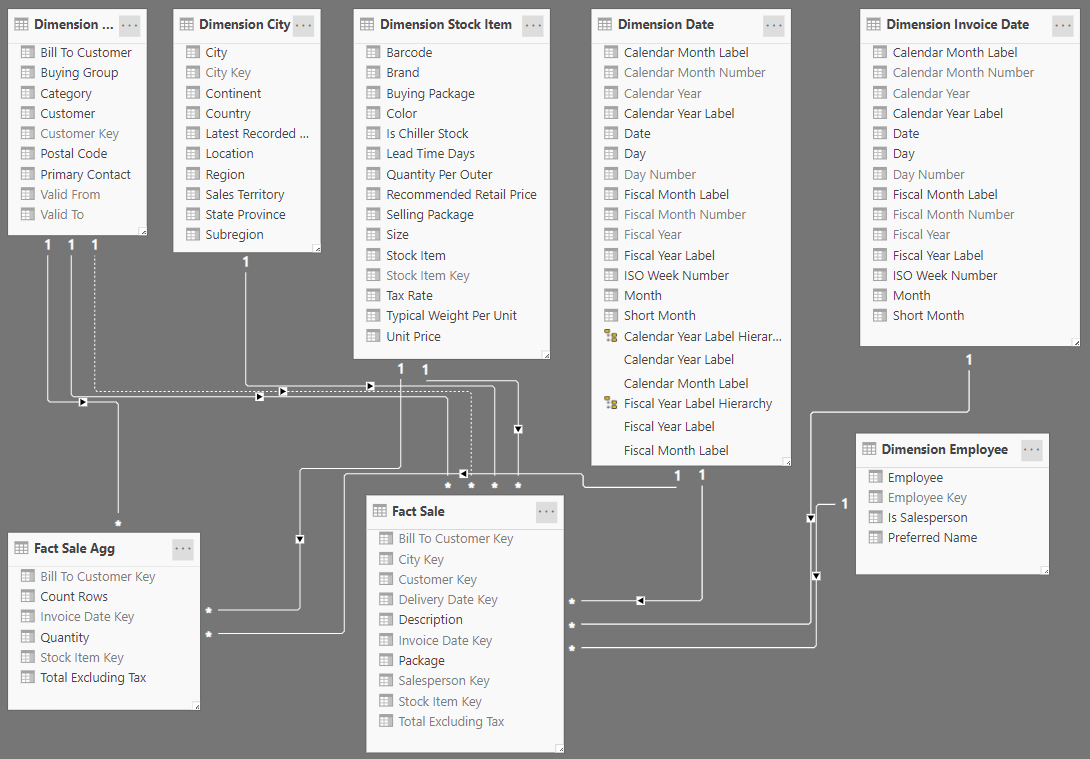
## How??

You use [IsFiltered](https://dax.guide/isfiltered/). Sometimes, you can use [IsCrossFiltered](https://dax.guide/iscrossfiltered/). That’s all there is to it. Move along.

…

Ok, there is a *little* more to it than just that.

Just like a UI Aggregation, to set up a Manual Aggregation, you need a data model with two tables about the same facts. For this article, we’re going to be using the [Wide World Importers](https://docs.microsoft.com/en-us/sql/samples/wide-world-importers-what-is?view=sql-server-2017) sample Data Warehouse available from Microsoft. Not the OLTP DB. I don’t know anything about OLTP. In this example, I’ve used the ‘Fact Sale’[[1]](https://d.docs.live.net/0081b09f311a432a/Blog%20Posts/Manual%20Aggregations/DAX%20Aggregations%20Blog%20Draft.docx" \l "_ftn1" \o ") table and all its related dimensions. I’ve created an aggregate table, ‘Fact Sale Agg’ that excludes Invoice Date, Delivery Customer (as distinct from Bill To Customer), and Employee ID as dimensions, as well as groups away the Sale specific rows.



Ok. So now that we have that, let's get back to what a UI Aggregation does. For each aggregation table, in order of precedence, it asks the question:

*Does the visual’s evaluation context include any dimensions that aren’t included in this table?*

So, let’s do that with DAX. For a single column, that might look like this:

   Measure =

 IF (

        ISFILTERED ( 'table'[Column] ),

        [Granular Table Measure],

        [Aggregated Table Measure]

 )

[Code beautified with Dax Formatter by SQLBI](https://www.daxformatter.com/)

How might we do this for multiple Columns? By using way too many OR() statements. Except, we don’t want to use [OR()](https://dax.guide/or/) because it only takes two arguments. Instead, we want to use the [Or Operator, ||](https://dax.guide/op/or/) because it takes *n* arguments. The reason we need *n* arguments is that IsFiltered()

*“Returns true when there are direct filters on the specified column.”*

So, we need to list every single column that filters the detail table that we’ve excluded from Aggregate Table. Like this:

Manual Aggregation Measure =

IF (

    ISFILTERED ( 'table 1'[Column1] )

        || ISFILTERED ( 'table 1'[Column 2] )

        || ISFILTERED ( 'table 2'[Column 3] )

        ...

        || ISFILTERED ( 'table n'[Column m] ),

    [Detail Table Measure],

    [Aggregated Table Measure]

)

[Code beautified with Dax Formatter by SQLBI](https://www.daxformatter.com/)

If you want to use multiple Agg Tables, just use SWITCH instead of IF, but I’ll leave that to you. For our sample data set, this becomes:

Total Sales DAX Agg =

IF (

    ISFILTERED ( 'Dimension Employee'[Employee] )

        || ISFILTERED ( 'Dimension Employee'[Employee Key] )

        || ISFILTERED ( 'Dimension Employee'[Employee] )

        || ISFILTERED ( 'Dimension Employee'[Is Salesperson] )

        || ISFILTERED ( 'Dimension Employee'[Preferred Name] )

        || ISFILTERED ( 'Dimension Invoice Date'[Date] )

        || ISFILTERED ( 'Dimension Invoice Date'[Calendar Month Label] )

        || ISFILTERED ( 'Dimension Invoice Date'[Calendar Year Label] )

        || ISFILTERED ( 'Dimension Invoice Date'[Fiscal Month Label] )

        || ISFILTERED ( 'Dimension Invoice Date'[Fiscal Year Label] )

        || ISFILTERED ( 'Dimension Invoice Date'[ISO Week Number] )

        || ISFILTERED ( 'Dimension Invoice Date'[Month] )

        || ISFILTERED ( 'Dimension Invoice Date'[Short Month] )

        || ISFILTERED ( 'Dimension Invoice Date'[Day] )

        || ISFILTERED ( 'Fact Sale'[Description] )

        || ISFILTERED ( 'Fact Sale'[Package] )

        || ISFILTERED ( 'Fact Sale'[Salesperson Key] )

        || ISFILTERED ( 'Fact Sale'[Delivery Date Key] ),

    [Total Sales Detail],

    [Total Sales Agg]

)

[Code beautified with Dax Formatter by SQLBI](https://www.daxformatter.com/)

What a mess of code. And that’s with a small number of excluded dimensions.

Can we simplify it? Maybe. There is no equivalent IsTableFiltered() function which returns true if any of the columns in the table are explicitly filtered. We could compare the number of rows in COUNTROWS(Table) with CALCULATE(COUNTROWS(Table),ALL(Table)). I haven’t tested it, but that doesn’t sound very fast.

We might also be able to use IsCrossFiltered(). However, this is where the big caveat comes in. IsCrossFiltered() “Returns TRUE when any column of the table specified *or another column a related table* is being filtered. Otherwise returns FALSE.” We’re going to have to think hard any bidirectional relationships or measures that invoke the Crossfilter() function involved, or it’s going to mess us up. I haven’t tested this either.

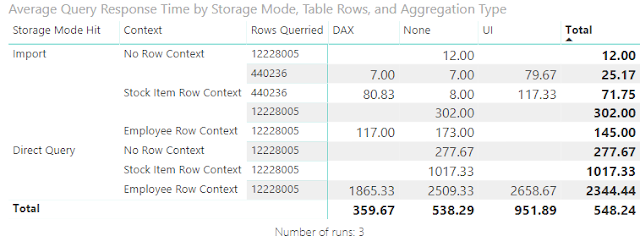
For now, at least, we’ve got a whole mess of Dax. Remind me why this is a good idea?

## Should???

So, let’s talk about whether this should this be considered an acceptable solution. There is a UI function that is a lot simpler to use and maintain with much of the same functionality. And most of the functionality gap that exists today is temporary. Do we even know if this thing is faster than just not bothering?

The main downside is complexity. You have to set up a measure with dozens of lines for every fact in your Agg table. Also, while I haven't tested this explicitly, my understanding is IF IsFiltered is Formula Engine territory, so you aren't going to want to nest your Manual Agg function inside an iterator. If you have any measures using sumx or the like, you'll want to setup separate Manual Aggregation measures for them too.

Measuring performance is more complicated. Your mileage may vary considerably. For this article, I tested the query times of SUM('Fact Sale'[Total Excluding Tax]). I performed my test in three filter contexts and repeated the test 3 times (more but I only saved the results 3 times, other runs were similar). I only tested relationship-based UI aggregations and I didn’t test DQ for the Agg table. The direct query source was the [World Wide Importers DW](https://docs.microsoft.com/en-us/sql/samples/wide-world-importers-dw-install-configure?view=sql-server-2017) sample with the [Large Sale table populated](https://docs.microsoft.com/en-us/sql/samples/wide-world-importers-generate-data?view=sql-server-2017#generate-data-in-wideworldimportersdw-for-performance-testing) running on a dedicated P1 Azure SQL DB with no other users. The aggregated table had 27x fewer rows than the unaggregated table.  The employee row context is not included in the agg table granularity and so isn’t able to generate an agg hit. Complete results along with a copy of the .pbix are available in the [supplemental materials](https://1drv.ms/f/s!AipDGjGfsIEA7TZ5EcyDfeZrcjur).

[](https://3.bp.blogspot.com/-8U-IC7ZZvog/XCh1I19ylYI/AAAAAAAAeDA/VoUgjPa4dgoMT0lLC2ohAOhC3_uqDQBWgCLcBGAs/s1600/Testing%2BSummary.png)

* Both types of Aggregations have an overhead over just querying an imported table.
* Both Aggregation types are significantly faster than direct query against the large ‘Fact Sale’ table.
* Performance over an imported but unaggregated ‘Fact Sale’ was better but not wildly better. However, 12 million rows is a pedestrian fact table and a simple sum the simplest of measures. Increased complexity will favor the Agg pattern.
* In my tests, Manual Aggregations outperformed UI Aggregations. However, I don’t think this data is even close to enough to say that this is usually, or even often the case. Once again, my tests are very simple compared to all the cases you’re likely to encounter. None the less, it is very interesting.

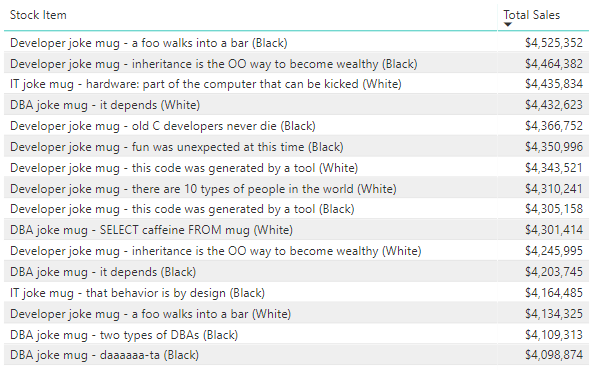
A Manual Aggregation does not result in a slow query. They definitely improve query times over not setting up any aggregation. Power BI is very fast at determining row context.

Overall, If you're concerned about performance. I would recommend using this technique only if UI Aggregations are not a solution to your problem. If you're trying to control security by granularity, then Manual Aggregations may be just the thing (at least until more details about how UI Aggs and RLS work together is available.)

# Part 2: Row Level Security with Delegate Access and Manual Aggregations

Now that we’ve established the why and how of manual aggregations (link), let’s dive into applications. Specifically, how we might pair this with Row Level Security. Or at least how I *did*pair it with RLS. This post assumes you have read part 1. You have been warned.

Let’s talk about World Wide Importer. Not the database, but the business. They sell stuff. From looking at the database,  it looks like they mostly sell coffee mugs to SQL Devs. Somehow they employ over 100 salespeople. They also employ a bunch of people to move them around the warehouse. All of these employees are listed in our model in 'Dimension Employee'. 'Fact Sale' rows are related to salespeople, and other fact tables in the model are related to warehouse staff.

[](https://2.bp.blogspot.com/-_SXLGm83y9Q/XCiDpk4XuwI/AAAAAAAAeDY/Gykgux5f1wsE7Cxvskhm5ItNs9qgyZ6gwCLcBGAs/s1600/WWI%2BCateloge.png)

We need to embellish a little to arrive at the reporting challenge I want to talk about.

First, assume that World Wide Importers pays their Sales People a sales commission or revenue-based incentive (like Bing!). HR doesn't like it when people know what their peers make, so sales data by employee is confidential.

With over 100 salespeople, they are going to have sales management, analysts, and operations staff to help run the show and set sales targets. Neither Sales Operations or Sales Management appear in the data warehouse because they don’t interact with the CRM or Warehouse applications that feed the OLTP system, so they aren’t associated with any records in the data. But they are going to need to see reporting on employee performance. However, HR still doesn't want sales managers knowing how their peers are getting paid.

Next, we’ll assume that we have some operations staff that is assigned directly to a specific sales team rather than HQ (like me!) and that this might be distinct from the company’s actual organizational structure. As a result, their access to employee sales data from needs to be limited to just the employees they are responsible for. We can pull org structure out of the HR database, but this kind of conditional access is going to need something more...custom.

Finally, we'll assume that we want to provide a single data model that can be accessed by the whole company and we want as many dimensions as possible to be available to everyone. We want to make sure that even Sales Managers can navigate the model using Analyze in Excel.

You know: Data Democratization; Single Source of Truth; Scalable Solutions; Clicky-Clicky Draggy-Droppy; Really want to justify playing with premium only features. Obvious stuff like that.

To recap, we want to design a data model security scheme that protects the privacy of people’s bonuses while also providing visibility into the performance of the business for all employees.

A.      A sales person can see their own revenue but not their peers.

B.      Sales managers can see the performance of their team but not their peers.

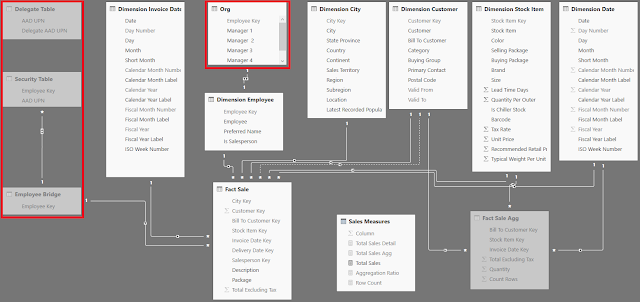
C.      Operations staff can see the performance of the team to which they are assigned.

D.      Other employees can't see any employee performance data.

E.       Everyone can see sales across non-employee dimensions.

F.       The user experience is seamless: measures are fast and you don’t need to know which version of sales to use for which dimensions.

Where we're going... we need to create some more tables.

[](https://3.bp.blogspot.com/-LEqAwPu88Es/XCh9oczX7vI/AAAAAAAAeDM/saHVK29klFUrSVf8MbbFVgMxdWetgU6CwCLcBGAs/s1600/RLS%2BExample%2BModel.png)

1. A leveled org table that relates Sales Managers to Employees. Manager(s) could also be dimensions of your employee table, but this depends on your Data Warehouse. I’m not modifying anything from the sample DW for my example, so we’ll just use an outrigger table  with a 1:1 relationship to Employee Key.
2. A flat employee security table. This is the org table unpivoted plus everyone listed as their own manager. Also, this table needs employees’ AAD User Principal Name.
3. A disconnected delegate access table that relates Operations staff with Managers. Also needs User Principal Names. By using a separate table, you can easily maintain it in excel or on your DB without having to get this to conform to any existing process. Unless one exists that works for you. Then use that.
4. The relationship between the security table and the fact table is many to many, so we have an employee bridge table.

It gets more complicated if you have multiple sales roles and a multi-level ragged hierarchy of sales managers (like Bing!) but let’s not go there today.

Thankfully, now that we’ve created a good data model, the DAX is very straightforward. This is always preferred.

We can create a single role with one simple DAX filter applied to the Security table to accomplish goals A-D above.

OR (

    [AAD UPN] = USERPRINCIPALNAME (),

    [AAD UPN]

        = LOOKUPVALUE (

            'Delegate Table'[Delegate AAD UPN],

            [AAD UPN], USERPRINCIPALNAME ()

        )

)

[Code beautified with Dax Formatter by SQLBI](https://www.daxformatter.com/)

This will filter Sale Fact down to rows where the user is listed in the security table, or where the person for whom they have delegate access is listed. Warehouse staff don’t appear in the fact table, so they don’t see anything. Ops staff have to be listed in the delegates table. That covers A-D. To accomplish E & F, we’re going to use a Manual Aggregation.

We never want to include a security table in the manual agg measure. This is because those columns are *always* filtered by our security rule, and thus would always cause us to hit our detail table if the participated This means that the security table *must* be a hidden table that is not used in any report visuals.

To finish this off, all we need to do is set up our DAX Measures for Total Sales.

Total Sales Detail =

SUM ( 'Fact Sale'[Total Excluding Tax] )

Total Sales Agg =

SUM ( 'Fact Sale Agg'[Total Excluding Tax] )

Total Sales =

IF (

    ISFILTERED ( 'Dimension Employee'[Employee] )

        || ISFILTERED ( 'Dimension Employee'[Employee Key] )

        || ISFILTERED ( 'Dimension Employee'[Employee] )

        || ISFILTERED ( 'Dimension Employee'[Is Salesperson] )

        || ISFILTERED ( Org[Manager 1] )

        || ISFILTERED ( Org[Manager 2] )

        || ISFILTERED ( Org[Manager 3] )

        || ISFILTERED ( Org[Manager 4] )

        || ISFILTERED ( Org[Employee Key] )

        || ISFILTERED ( 'Dimension Invoice Date'[Date] )

        || ISFILTERED ( 'Dimension Invoice Date'[Calendar Month Label] )

        || ISFILTERED ( 'Dimension Invoice Date'[Calendar Year Label] )

        || ISFILTERED ( 'Dimension Invoice Date'[Fiscal Month Label] )

        || ISFILTERED ( 'Dimension Invoice Date'[Fiscal Year Label] )

        || ISFILTERED ( 'Dimension Invoice Date'[ISO Week Number] )

        || ISFILTERED ( 'Dimension Invoice Date'[Month] )

        || ISFILTERED ( 'Dimension Invoice Date'[Short Month] )

        || ISFILTERED ( 'Dimension Invoice Date'[Day] )

        || ISFILTERED ( 'Fact Sale'[Description] )

        || ISFILTERED ( 'Fact Sale'[Package] )

        || ISFILTERED ( 'Fact Sale'[Salesperson Key] )

        || ISFILTERED ( 'Fact Sale'[Delivery Date Key] ),

    [Total Sales Detail],

    [Total Sales Agg]

)

[Code beautified with Dax Formatter by SQLBI](https://www.daxformatter.com/)

Now we just hide everything but [Total Sales] so that all the user sees is that measure. Now they are seamlessly directed to the right granularity for their query and when they filter by the employee or org table, they only see the Sales Rows that they are linked to.

You may later decide to include some columns of the employee in your aggregate fact table. For example, your sales regions and sales manager alignment may be a bit messy (like Bing!) and so you might want to expose revenue for a higher-level manager to the report audience regardless of role.

If you wanted to get crazy you can even add another, totally different, RLS rule to your Agg table.

The sample .pbix files are available [here](https://1drv.ms/f/s!AipDGjGfsIEA7TZ5EcyDfeZrcjur).

Good luck.