Article
Evgeniy Potapov · Aug 16, 2022 7m read

How to make drill down from IRIS DB to PowerBI via Adaptive Analytics

One of the most important tasks of dashboards is, on the one hand, to help you perceive data in an aggregated form, and, on the other hand, not to lose the depth of immersion in the data if you need this. One of the tools that help us achieve this result is drill down. It enables us to display several hierarchical levels of data, from aggregated to detailed.
Our task is to make an analytical dashboard that will be able to drill down by time periods: from months to weeks and days in each chart. The stack that we will use for work consists of IRIS as a data storage system, Adaptive Analytics for data aggregation and building complex queries, and Power BI as a BI system. We use Adaptive Analytics as an intermediate system to turn the IRIS tabular database into an OLAP-like one. This is useful for many BI systems. It allows us to prepare analytical data for different BI systems in one place and can be used for various tasks.

I need to point out that for the convenience of using dashboards, we have set filters by dates (e.g. last 6 months), in order not to display data for all years at once (this will be important in the future).

Power BI has 2 ways to connect to OLAP data sources. The first method is a direct connection when in Power BI we see only measures and dimensions, and all calculations and queries occur on the database side. The second method requires uploading data by queries from an OLAP database to a Power BI file and storing this data in the form of tables. The Power BI provides a fairly powerful toolkit to aggregate table data. During the development stage, we had some fears which, subsequently, were justified. The thing is that a direct connection to our data set makes queries run long enough to cause discomfort for the end user due to waiting time. That is why we chose the option of storing data in a file since the results of aggregated queries did not take up much space.

If we need to drill down by day, we can no longer take full advantage of the aggregation capabilities of Adaptive Analytics. We have to transfer 30 times more data, and the calculations themselves become 30 times more difficult since now we need to aggregate the data 30 times a month for each day instead of only once a month. Accordingly, the question of performance arises since the regular updating of data should not take much time and resources. By the way, the Adaptive Analytics to Power BI connection has difficulties with automating data updates. Perhaps in the future, we will describe the solution to this problem in a separate article.

In order to get the best query performance, we needed to understand how measurements worked. There are two types of dimensions in Adaptive Analytics - actual and degenerate dimensions. Degenerate dimensions include all records from the columns bound to them, while not linking the tables to each other. Normal dimensions are based on one column of one table, and only unique values can be selected from it. Other tables can be linked to this dimension. Data for records that have no key in the dimension is simply ignored (for example, if the main table does not have a specific date, then data from related tables for this date is skipped in calculations since there is no such member in the dimension).
From a usability point of view, degenerate dimensions are more convenient because it is not possible to lose data or gain a relationship between cubes in a project that was not intended. However, from a performance point of view, the use of normal dimensions is preferable, and for our task, with hundreds of thousands of lines, it is simply necessary.

Initially, to solve problems with the fact that new data does not appear every day and there may be unique dates in different tables, which disappear when linked by a dimension from another table, we used degenerate dimensions. However, the use of the calendar (you can read about this solution (https://community.intersystems.com/post/showing-dates-missing-periods) in our previous article) allowed us to solve this problem differently, switch to more productive dimensions and get enough performance to work with drill down.

By having well-optimized cubes in Adaptive Analytics, we can make sure that updating data will not take too long. We query all the data we need into a table in Power BI and use the date, week, and month fields as a hierarchy for the charts. Thus, we get drill down for regular charts.

Another important tool in analytics is the running total. This is the summary of all the values of an entity throughout time. It allows you to look at the growth data not in absolute terms, but proportionally to the total mass. As an example: 1000 new subscribers on a YouTube channel per month - is it good or bad? If we have only 10,000 subscribers, then it is probably good. However, if we have a million, then we might conclude that the channel is becoming boring and needs some urgent changes.

For running total charts, we would also like to see a drill down to estimate growth over any given period of time. Power BI, apart from other options for creating quick measures, in addition, offers us to make a running total. It usually works well except for one thing, which I mentioned at the beginning of the article - we need to filter data by time period for display convenience. Since Power BI filters the data table instead of just hiding unnecessary parts of the visual element, we lose part of the data, and the running total is calculated only for the remaining values, which is not the right solution.
In our experience, there is no single correct solution for this problem. The first solution is to have a calculated running total in a table. You can pull it up from Adaptive Analytics, or you can calculate it in Power BI. YYou can find many articles and videos about calculations in Power BI. I do not want to repeat them. In Adaptive Analytics, you can create a calculated measure and specify the MDX-like syntax:

```
Sum([DataDimension].[DataDimension].CurrentMember.FirstChild : [DataDimension].[DataD
imension].CurrentMember ,[Measures].[some_measure])
```

where [DataDimension].[DataDimension] - the first 2 levels of your time dimension (the name of the dimension and the name of the hierarchy within the dimension). Mind that
CurrentMember refers to the lowest element of the hierarchy, and all calculations will go through it.
[Measures].[somemeasure] - the usual measure for which you want to get a running total.

Once you have the data in the table, you can display the maximum of that column for each date. Since all calculations have been made, and the number has already been recorded in the table, the filter by time period does not interfere with the results.

Yet, this solution has a problem that appears if our data is used by another category, and we want to filter the chart by it. For instance, we have the ability to track the source of channel subscribers (let's say we have multiple advertising channels), and we want a running total with the ability to filter by these sources. In this case, the table will have two values: a user and a related source. So, when filtering the table by source in the running total column, we will get numbers in the form of 1,3,5,8 ... instead of 1,2,3,4 ..., which will result in a wrong summary.

To deal with this issue you will have to use the calculated cumulative total. As we remember, its main problem is filtering the data table. You can turn off this function, but then the chart will include all the dates from the database. I am not going to describe all the possible ways to solve this problem. Instead, I will give you the most adequately working one from our experience.

Since the filter is set to a common calendar for all tables, we can use the date from the table itself when calculating cumulative total. If we do that, all calculations will be performed correctly, but only for the dates in the table. Then only the dates from the filtered calendar will be displayed, and we will get a chart that interacts with the filter normally. However, there is one drawback - we will be able to see only the dates that were present in the table. If you do not have missing dates in the table, or it is not critical for you, you can use this solution. The formula for DAX looks like this:

```
Total_measure = Calculate(
            [Measure],
            ALL('Calendar'),
            'Table_name'[date_dimension_name] <= MAX('Table_name'[date_dimension_name
])
```

```
        )
```

If you need to see all dates (for those that had no new data, the value from the previous ones will be displayed), then you can use a more complicated formula. According to the original formula, we get the desired value for each of the dates in the filtered interval. Now we need to go through all the dates from the calendar and for each of them get the value from the closest date present in the calculation. Here is a DAX formula for this:

```
Total_measure = Calculate(
        Calculate(
            [Measure],
            ALL('Calendar'),
            'Table_name'[date_dimension_name] <= MAX('Table_name'[date_dimension_name
])
        ),
    ALL('Calendar'),
    'Calendar'[Date] <= MAX('Calendar'[Date])
)
```

This solution also has a problem. If there is no data for a certain period of time, the filter will not see it. In this case, if from the last 6 months there is no new data for the first one, then only 5 months will be displayed on the chart.

As you can see, each of the solutions works better for its type of task, and it ' s up to you to decide which one you should use.

#Adaptive Analytics #Analytics #MDX #InterSystems IRIS

Source URL:https://community.intersystems.com/post/how-make-drill-down-iris-db-powerbi-adaptive-analytics