Time Shifted Metrics:

How we make a staple of Business Intelligence work on Hadoop

TIPS FROM ATSCALE: Period-over-period comparisons are among the most common calculations in the world of Business Intelligence. They are useful in understanding growth, identifying trends, and evaluating performance. In this tech tip we will discuss how AtScale's easy to use modeling environment enables the creation of period-over-period metrics using time-shifted data sets.

As more and more enterprises look to use Hadoop to support Business Intelligence use cases, the demand for a traditional OLAP experience grows with it. While Hadoop excels and low-cost distributed storage and parallel data processing, interactive support for BI-style queries remains a challenge. Additionally, multi-dimensional queries often demand complex OLAP-style calculations and functions. In this tech tip we will share how AtScale's easy to use modeling environment enables the creation of period-over-period metrics using time-shifted data sets.

Key Components of Time Shifted Metrics

In many typical business analyses or applications, it is common to look at period-over-period comparisons to assess growth, identify trends, and evaluate performance. For example

- How did my sales for Product X this week compare to sales for the same week and product last year?
- What was the growth of sales in Region A when compared to last quarter?
- What does week-over-week web traffic growth look like for my new landing page?

The AtScale approach to security takes these requirements into consideration, and as such delivers the security controls that are described in more detail in the following sections.

Business Users Want Measures and Dimensions, not Tables, Columns and SQL

While the concept is relatively simple, creating period-over-period measures in a traditional SQL-based BI tools is challenge, as this discussion thread highlights. Period-over-period comparisons can also be done in hand-written SQL, such as this query:

```
SELECT InvoiceNum, InvoiceDate, T1.CustAcct, This_Year_Sales,
This_Year_Profit, Last_Year_Sales, Last_Year_Profit
FROM (
     SELECT
       InvoiceNum,
       InvoiceDate,
       Custid,
       TotSales as This_Year_Sales,
       PubProfit as This_Year_Profit,
       0.00 as Last_Year_Sales,
       0.00 as Last_Year_Profit
     FROM arinvhed
     where InvoiceDate Between
DATEADD(wk,DATEDIFF(wk,0,GETDATE()),0)  and GETDATE()

     UNION ALL

     SELECT
       InvoiceNum,
       InvoiceDate,
       Custid,
       0.00 as This_Year_Sales,
       0.00 as This_Year_Profit,
       TotSales as Last_Year_Sales,
       PubProfit as Last_Year_Profit
     FROM arinvhed
     where InvoiceDate Between dateadd(yy,-
1,DATEADD(wk,DATEDIFF(wk,0,GETDATE()),0))  and dateadd(yy,-
1,GETDATE())
     ) T
Left Outer Join arcusts T1 On T.CustID = T1.CustID
```

But the challenge of this approach is that the computation is difficult to share and standardize across what may be a number of BI users and a number of BI-client tools. Another option is to use an MDX-compliant OLAP server to support MDX calculations like the one below… but not all BI clients support MDX.

```
WITH
MEMBER [Measures].[Prior Period Reseller Sales Amount] AS
   (
       ParallelPeriod(
           [Date].[Calendar].[Calendar Year],
           1,
           [Date].[Calendar].CurrentMember
           ) ,
       [Measures].[Reseller Sales Amount]
       )
```

```
    ,FORMAT="Currency"
MEMBER [Measures].[Prior Period Growth] AS
    (
        ([Measures].[Reseller Sales Amount])-
            ([Measures].[Prior Period Reseller Sales Amount])
        ) /
        ([Measures].[Prior Period Reseller Sales Amount])
    ,FORMAT="Percent"
SELECT
    {
        ([Measures].[Reseller Sales Amount]) ,
        ([Measures].[Prior Period Reseller Sales Amount]),
        ([Measures].[Prior Period Growth])
        } ON COLUMNS,
     {
        Descendants(
            [Date].[Calendar].[Calendar Year].[CY 2003],
            [Date].[Calendar].[Month],
            SELF
            )
         } ON ROWS
FROM [Step-by-Step]
```

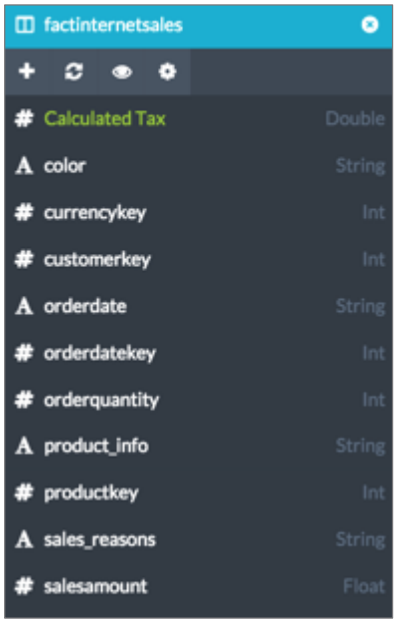
With AtScale, it's possible to create a single semantic definition of a time-shifted metric that can be easily consumed by Tableau users (and any other BI tool that talks SQL) while at the same time exposing the same metric to MDX clients, which means that virtually any data visualization client can take advantage of this advanced functionality.

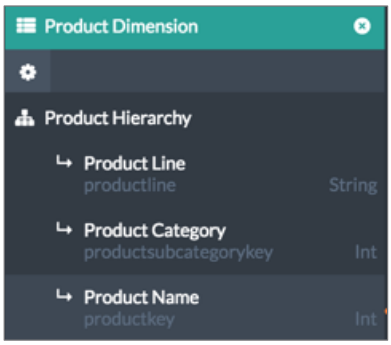
The following sections describe how to create and subsequently query period-over- period measures using the AtScale Intelligence Platform.

The Basics

The example to the right is based on a slightly modified version of Microsoft's Adventureworks Database. This data has been loaded into a Hadoop cluster and modeled using Hive. Our fact table for this analysis is the factinternetsales table, as shown here. For purposes of this document we will focus on creating 4 metrics:

- Sales Amount (sum of salesamount)
- Order Quantity (sum of orderquantity)
- Last Year Sales Amount
- Last Year Order Quantity

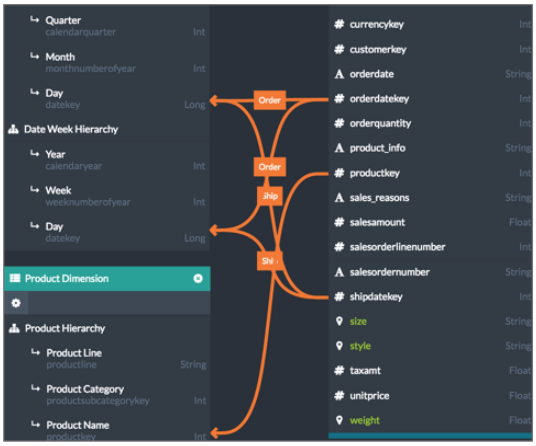


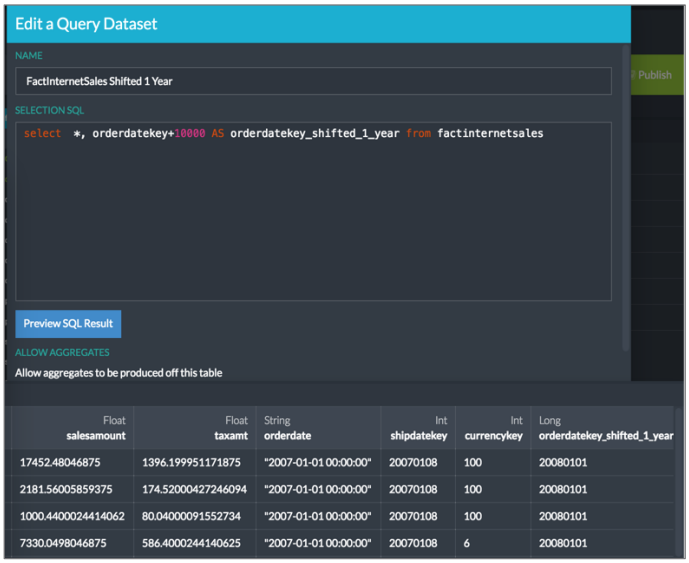


We also will use a Product Hierarchy to analyze our Order Quantity and Sales Amount metrics over time. Like any fully-functional OLAP server, AtScale provides rich support for hierarchies, including multiple hierarchies per dimension, secondary attributes and role-playing relationships. The product hierarchy for our example is shown here. For our sample analysis we are going to look at the performance of several products this year and compare their performance with the same period(s) of the previous year.

Creating Measures and Relationships

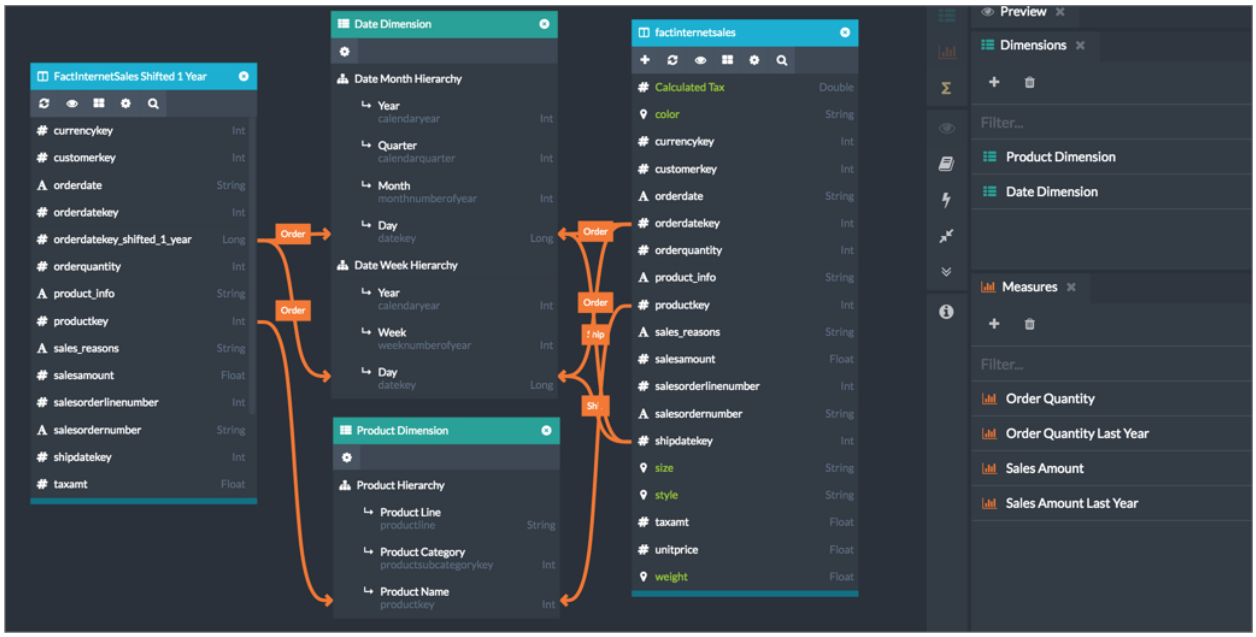
In this basic cube we've already created a relationship between our base fact table, `factinternetsales`, and our Date Dimension and Product Dimension, as shown to the right. Note the use of AtScale's role-playing functionality - our shared Date Dimension is related to both `orderdatekey` and `shipdatekey`, as evidenced by the role-playing prefixes that are attached to the relationship lines.





In our standard Sales Insights Cube we already have two measures - Sales Amount and Order Quantity - that have been created. But now I want to add two new measures that reflect the value of Sales Amount and Order Quantity 1 year ago. The first step in doing this is to create a "time-shifted" version of our `factinternetsales` table, using AtScale's SQL Dataset capability. In the image on the left you can see how I've shifted our standard `orderdatekey` column by one year, creating `orderdatekey_shifted_1_year`.

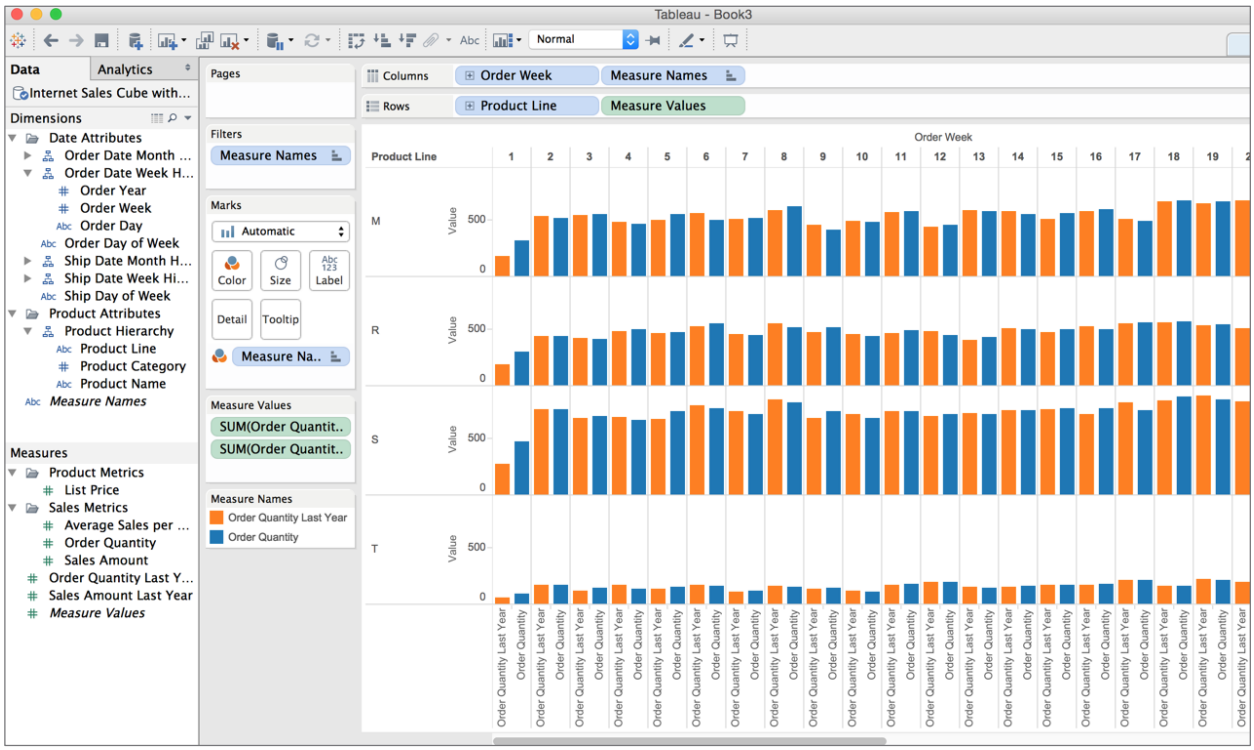
The final step in this process is now to create the appropriate relationships and measures in my AtScale Virtual Cube, using this new data to create a *multi-fact model*. As the screenshot below shows, I've now created a shared set of dimension relationships between my two fact tables, joining both `orderdatekey` and `orderdatekey_shifted_1_year` to my Date Dimension. Additionally, I've created two new metrics to represent Sales Amount Last Year and Order Quantity Last Year.



Analyzing the Data

Now that the measures and dimensions for this cube have been created, they can be queried using a data visualization front end like Tableau or Microsoft Excel.

In Tableau, the AtScale cube can be accessed using an automatically-created TDS (Tableau Data Source) file that includes the hierarchical structure as designed in the cube. Once open, it's simple to create the following report by dragging and dropping our Order Quantity and Order Quantity Last Year metrics and the Week level of the data hierarchy onto the canvas. Additionally, I can create separate reports for each of my Product Lines by dragging this onto the Rows shelf. This report now allows me to look at the weekly performance of my key product lines, compared to the same week last year.



To confirm the validity of the results, we look at the values of our current and previous measures at a yearly breakdown. As is shown below, the values for the Last Year Sales Amount for 2007 exactly matches the Sales Amount for 2006, confirming that our approach has appropriately time-shifted our metrics.

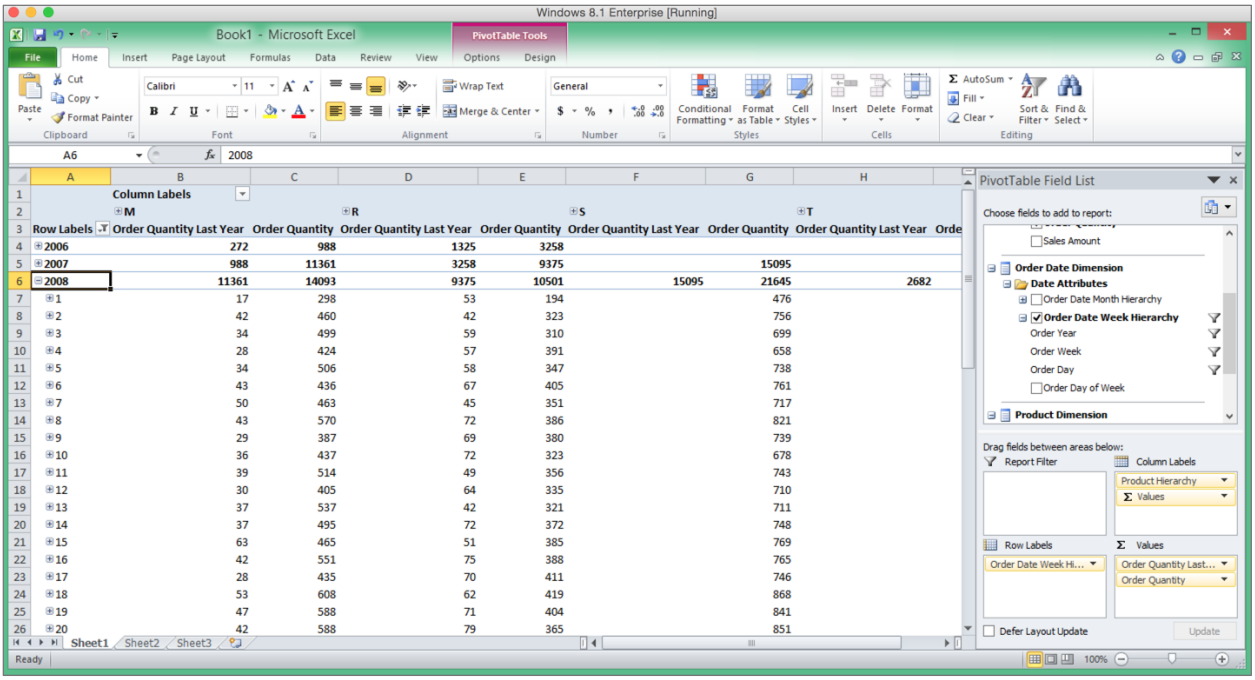
Order Year	Order Quantity	Order Quantity Last Year	Sales Amount	Sales Amount Last Year
2006	4,246	1,597	10,276,314	5,135,542
2007	38,513	4,246	15,551,189	10,276,314
2008	50,811	38,513	15,230,723	15,551,189

It's worthwhile at this point to comment on the simplicity of doing this analysis in Tableau versus the complexity of the query that needs to be executed on the underlying raw data tables in Hadoop. Because AtScale abstracts away the complexity of this query, this means that Tableau (or any other system or person querying AtScale in SQL) can submit a very simple query, like this one:

Without AtScale, this query would need to be written, by hand and submitted to Hadoop as follows (this is the query that AtScale automatically generates using its knowledge of the underlying Hadoop data structures):

```
SELECT
  t_11.c_1 order_week,
  t_11.c_2 product_line,
  MIN(t_11.c_3) sum_m_orderquantity_sum_ok,
  MIN(t_11.c_4) sum_orderquantity1_ok
FROM
(
     SELECT
        dimdate_t6.weeknumberofyear c_1,
        dimproduct_t7.productline c_2,
        NULL c_3,
        SUM(factinternetsales_t5.orderquantity) c_4
     FROM
        as_adventure.factinternetsales factinternetsales_t5
     JOIN
        as_adventure.dimdate dimdate_t6
     ON
        factinternetsales_t5.orderdatekey = dimdate_t6.datekey
     JOIN
        as_adventure.dimproduct dimproduct_t7
     ON
        factinternetsales_t5.productkey = dimproduct_t7.productkey
     GROUP BY
        1,
        2
  UNION ALL
     SELECT
        dimdate_t9.weeknumberofyear c_1,
        dimproduct_t10.productline c_2,
        SUM(t_8.orderquantity) c_3,
        NULL c_4
     FROM
     (
select  *, orderdatekey+10000 AS orderdatekey_shifted_1_year from
factinternetsales
     ) t_8
     JOIN
        as_adventure.dimdate dimdate_t9
     ON
        t_8.orderdatekey_shifted_1_year = dimdate_t9.datekey
     JOIN
        as_adventure.dimproduct dimproduct_t10
     ON
        t_8.productkey = dimproduct_t10.productkey
     GROUP BY
        1,
        2
) t_11
GROUP BY
  1,
  2
```

In addition to this data being accessible via a SQL query (generated by Tableau, by hand, or by another SQL client) this exact same cube model is accessible by clients that utilize the MDX query language as well. For example, using an MDX client such as Microsoft Excel, we can access the exact same cube in AtScale, and arrive at the same results.



Row Labels	M Order Quantity Last Year	M Order Quantity	R Order Quantity Last Year	R Order Quantity	S Order Quantity Last Year	S Order Quantity	T Order Quantity Last Year	Orde
2006	272	988	1325	3258				
2007	988	11361	3258	9375		15095		
2008	11361	14093	9375	10501	15095	21645	2682	
1	17	298	53	194		476		
2	42	460	42	323		756		
3	34	499	59	310		699		
4	28	424	57	391		658		
5	34	506	58	347		738		
6	43	436	67	405		761		
7	50	463	45	351		717		
8	43	570	72	386		821		
9	29	387	69	380		739		
10	36	437	72	323		678		
11	39	514	49	356		743		
12	30	405	64	335		710		
13	37	537	42	321		711		
14	37	495	72	372		748		
15	63	465	51	385		769		
16	42	551	75	388		765		
17	28	435	70	411		746		
18	53	608	62	419		868		
19	47	588	71	404		841		
20	42	588	79	365		851		

In this example, Excel is connecting to the AtScale cube as if it were a SQL Server Analysis Services data provider (using the XMLA protocol), and it's now possible to interact directly with the data on the underlying Hadoop deployment in live query mode. Because the underlying semantic data model is shared regardless of the dialect used (SQL or MDX) the results across all tools are identical.

Summary

As the simple example in this post shows, AtScale's Virtual Cube with multi-fact model support enables common analytical use cases, including period-over-period comparisons. The AtScale Intelligence Platform supports these concepts directly on top of data sets that are stored in a Hadoop Hive Warehouse, effectively transforming a Hadoop cluster into a Scale-Out OLAP Server. In addition to providing an intuitive and simple design process for AtScale Virtual Cubes, the AtScale platform supports virtually any Business Intelligence or Data Visualization front-end by providing support for both SQL and MDX query languages against a shared, consistent semantic layer.