The Big Data Accelerator

Manual Partitioning on SQL Server

A Fully Worked Example

By

Version 0.01
1st January 2017
Empower Business Intelligence
info@empowerbi.com

Table of Contents

1. EXECUTIVE SUMMARY

Welcome to our White Paper providing a fully worked example of how to Manually Partition Fact Tables using MicroStrategy and SQL Server Standard Edition.

In this White Paper we provide detailed information for your Data Warehouse Support Staff on how to implement technical improvements to your data warehouse environment.

If your company is using SQL Server 2008R2 and has implemented database level partitioning this White Paper may be able to help your Data Warehouse Support Staff save money on your upgrade to SQL Server 2012.

You would save this money by upgrading to SQL Server Standard Edition rather than Enterprise Edition.

For many companies moving to Standard Edition rather than Enterprise Edition represents a significant saving.

If your company did not implement any form of Partitioning for your Data Warehouse this White Paper will show your Data Warehouse Support Staff how to:

- Improve your Data Warehouse Query Performance
- Improve Your ETL Performance so your data is ready sooner each day
- Reduce the time taken for DBAs to support the database which will reduce support costs.

By using the information freely available in this document your Data Warehouse Support Staff can make your Data Warehouse go faster without buying more hardware. It will help your Data Warehouse Support Staff make better use of the hardware and software you have already paid for.

At the same time you Data Warehouse Support Staff will be able to save money on support costs by reducing the time and effort required to support your Data Warehouse.

“Go Faster” + “Save Money” are two compelling reasons why your Data Warehouse Support Staff should take a close look at the information we are freely providing to you.

Whether your company uses our ideas or not?

We are pleased that you are taking the time to review this information.

We would be happy to answer any questions you have about how to make your SQL Server Data Warehouse go faster and how to reduce your costs of support.

We look forward to hearing from you.

Best Regards

Iulian Lixandru
CEO
Empower BI
iulian@empowerbi.com

2. AUDIENCE

Welcome to our white paper on BIDA Partitioned Fact Tables Demonstration.

We hope you will like this white paper.

You should read this white paper if you are most of the following:

- An SQL Server user for your data warehouse.
- A MicroStrategy customer.
- A data warehouse Project Manager who would like to know if Manual Partitioning is for you.
- A data warehouse DBA who would like to know if Manual Partitioning is for you.
- Concerned at the licensing differences between SQL Server Standard Edition and Enterprise Edition.

If you are most, or all, of those things we believe you will find this White Paper valuable.

3. PURPOSE

With the introduction of SQL Server 2005 Microsoft introduced Database Level Partitioning. This was a long awaited feature for the SQL Server data warehousing community. Until this version of SQL Server SQL Server DBAs had to implement what was called Manual Partitioning. This included extra DBA work.

MicroSoft was one of the last major database vendors to introduce database level partitioning and so it was an eagerly awaited feature for all those SQL Server DBAs who wanted to move to a database supported partitioning strategy for their Data Warehouse.

In SQL Server 2008 and 2008R2 the database level partitioning feature was further enhanced.

During this time the database level partitioning feature was only available in the Enterprise Edition of SQL server.

This was not considered an issue as the price difference between lower editions such as BI Edition or Standard Edition was not so great.

When MicroSoft released SQL Server 2012 the licensing options were changed. SQL Server Enterprise Edition was only available on a per core pricing license and no longer available on a Client Access License.

For companies that had built their data warehouses on SQL Server 2008R2 Enterprise Edition using the database level Partitioning Feature this meant steep upgrade pricing.

Even though support for SQL Server 2008R2 ended in 2014 there is limited support until 2019. There are still quite a few companies out there running SQL Server 2008R2 using the partitioning feature who do not want to pay the price of SQL Server 2012/14/16 Enterprise Edition.

Further, there are many companies out there who are using SQL Server Standard Edition (2012/14/16) for their data warehouses who have not implemented any form of partitioning though lack of knowledge or lack of understanding of how to do so.

The Purpose of this White Paper is to show you:

1. The major benefits of Manual Partitioning using MicroStrategys Partitioned Tables Feature.
2. How, exactly, to implement Manual Partitioning using SQL Server 2014 and MicroStrategy.
3. The example code generated when such Manual Partitioning is implemented.

4. INTRODUCTION TO PARTITIONING

"Partitioning is the division of a large table into smaller tables. You often implement partitioning in a data warehouse to improve query performance by reducing the number of records that queries much scan to retrieve a result set. You can also use partitioning to decrease the amount of time necessary to data into data warehouse tables and perform batch processing.

*Databases differ dramatically in the size of the data files and physical tables they can manage effectively. Partitioning support varies by database. Most database vendors today provide some support for partitioning at the database level. Regardless, the use of some partitioning strategy is essential in designing a manageable data warehouse. Like all data warehouse tuning techniques, you should periodically re-evaluate your partitioning strategy." **

* From the MicroStrategy Advanced Project Design Manual Page 201.

Simply put the main benefits of partitioning fact tables in a data warehouse environment are:

1. Reduce the execution time of both queries and maintenance activities
2. Reduce the resources required to perform both queries and maintenance activities.
3. Reduce the people time required to manage the data warehouse tables.

As we all know, anything that reduces resources required, reduces run times, and reduces the amount of people time needed to maintain a data warehouse is something that should be considered.

Now. It is not the purpose of this White Paper to explain what Partitioning is in detail. There are a great deal of resources on blog posts on the web and in existing documents which explains why fact tables should be partitioned in a data warehouse.

As the excerpt says it is the process of splitting up large tables in to smaller tables to make the overall data warehouse more manageable. The various database vendors take different approaches to how this splitting of data is implemented. However, the result is always similar. The large tables are "some how" split to improve the manageability of the overall data warehouse.

We would hope that the fact that every major data warehouse database vendor includes partitioning features, and recommends they are used, would indicate to you that partitioning is something that you should do in your data warehouse.

In our experience attempting to build a data warehouse without partitioning the larger fact tables creates a vast array of data management problems which are best avoided.

IBM first introduced the ability to partition tables for data warehouses way back in the early 90s. This should give you an idea of how necessary partitioning, in some way, is for a data warehouse.

As the MicroStrategy Manual states:

"*the use of some partitioning strategy is essential in designing a manageable data warehouse.*"

If you are not performing some form of partitioning and your data warehouse has any significant volumes in it, such as 100 million rows plus in fact tables, you should be partitioning your data.

You can use SQL Server Database Partitioning in Enterprise Edition. However, the license fees for the Enterprise Edition are significant and they may be beyond your budget. It is a great feature which is now very mature having been available for 10 years. It is just that MicroSoft wants to be well compensated for the availability of this feature.

There is a way to implement the same benefits without the same cost when using MicroStrategy.

From this point this paper will talk about Manual Partitioning as opposed to SQL Servers Database Partitioning Feature supported in Enterprise Edition.

4.1. The Benefits of Manual Partitioning

The main benefits of Manual Partitioning using SQL Server and MicroStrategy can be summarised as follows:

1. Reduced SQL Server Fees by using Standard Edition or BI Edition.
2. Generation of SQL code that is easier to understand.
3. Simpler Query Plans are Generated.
4. Increased reliability of SQL Server performance levels.

So let us address these one at a time.

Reduced SQL Server Fees by using Standard Edition or BI Edition

SQL Server Standard Edition and BI Edition are available as Client Access License Products.

This means that for smaller companies who have fewer users of their data warehouse licenses can be purchased on a per user basis. This pricing structure is usually better for smaller companies because more cores can be added to the SQL server without an increase in license fees.

The per core license fee structure is more suited to larger companies that have the money to spend on such license fees to improve performance for the user community.

For smaller companies that might have only 10-50 BI users on MicroStrategy the per user licensing strategy for SQL server is far preferable to the per core SQL Server Licensing model.

Generation of SQL code that is easier to understand

When using Manual Partitioning with MicroStrategy the MicroStrategy engine itself generates the SQL Code that accesses the different partitions of the tables. This means that by reading this SQL Code you can tell if the correct Partitions are being accessed.

This MicroStrategy generated SQL Code is much easier for the average SQL literate person to read to make sure that the query is doing what it is supposed to do. Even the average end user could review the SQL to make sure that the SQL being generated is accessing the correct partitions because they are visible in the SQL mode of any report.

When using Database Partitioning the SQL Generated, though it may look simpler, does not give the user any indication as to whether the correct partitions are being accessed. This can only be seen by experienced DBAs and only if they are given the SQL to analyse in DBA only tools.

The average end user of MicroStrategy has no ability to see how the database is answering the query and therefor any poorly performing report has to be sent to the DBAs. With manual partitioning the user might quickly browse the SQL generated and notice that s/he has missed a constraint and that all partitions are being accessed.

This visibility of the generated SQL means that the very first level of performance problem investigation can be done by the end user or SQL Literate support personnel and does not need to wait for the DBAs to make some time available.

Even though it is getting a little ahead of ourselves let us take 5 minutes to review the sample SQL that will be generated using MicroStrategy in this example. Do not be too concerned about how MicroStrategy manages to generated this SQL code. All will be explained in detail later in the document.

This SQL snippet generates the data for a very simple Sales Report for three days that occur in three different months. The pk_vm_day in statement lists the keys for the three days.

```
select distinct a11.PBTNAME  PBTNAME
from   dbo.vf_sale_txn_parts a11
       join   dbo.vm_day    a12
         on   (a11.partitioning_column = a12.partitioning_column)
where  a12.pk_vm_day in (10889, 10925, 10958)
```

The vf_sale_txn_parts table is a small table that needs to be created to tell MicroStrategy where it can find the actual partitions of data for the months involved. There is a “partitioning_column” that is on both the vf_sale_txn_parts table and the vm_day table. We use a separate column as a “partitioning_column” to give us complete control over the partitioning period. For example the partitioning period could be weekly, monthly, quarterly. It could be by the retail 4-5-4 type months. It could be in the Islamic calendar. However you decide to partition fact tables by time you will be able to break up the partitioning by day.

```
select a11.dk_vm_sale_date                pk_vm_day,
       max(a12.day_date)                  day_date,
       sum(a11.sale_extended_amount)      WJXBFS1
from   dbo.vf_sale_txn_1501               a11
       join   dbo.vm_day                  a12
       on     (a11.dk_vm_sale_date = a12.pk_vm_day)
where  a11.dk_vm_sale_date in (10889, 10925, 10958)
group by      a11.dk_vm_sale_date
union all
select a11.dk_vm_sale_date                pk_vm_day,
       max(a12.day_date)                  day_date,
       sum(a11.sale_extended_amount)      WJXBFS1
from   dbo.vf_sale_txn_1502               a11
       join   dbo.vm_day                  a12
       on     (a11.dk_vm_sale_date = a12.pk_vm_day)
where  a11.dk_vm_sale_date in (10889, 10925, 10958)
group by      a11.dk_vm_sale_date
union all
select a11.dk_vm_sale_date                pk_vm_day,
       max(a12.day_date)                  day_date,
       sum(a11.sale_extended_amount)      WJXBFS1
from   dbo.vf_sale_txn_1503               a11
       join   dbo.vm_day                  a12
       on     (a11.dk_vm_sale_date = a12.pk_vm_day)
where  a11.dk_vm_sale_date in (10889, 10925, 10958)
group by      a11.dk_vm_sale_date
```

The three days selected are in January, February and March 2015. The keys for these three days happen to be `dk_vm_sale_date in (10889, 10925, 10958)`. This means that the rows that must be retrieved for these three days must be in three different tables with the names **vf_sale_txn_1501/02/03.**

In our example we created three years worth of tables, 36 tables. Most organisations will have many more years of sales transactions than three years. We are just creating an example.

Clearly, if we split three years worth of transactions in to 36 tables, one per month, and we build indexes on those tables, the tables and indexes will be much smaller and faster to query than one large table with large indexes.

It will also be faster to back up only the recent tables and to maintain only the recent tables and indexes.

The “union all” brings back the results of each of the sub queries and joins them together to send them back to MicoStrategy for presentation on the report.

We hope that by showing you the example SQL you can get an early idea of the SQL we are trying to generate which will run faster than the similar query against one large sales fact table.

Simpler Query Plans are Generated

One of the big issues of database level partitioning is the complexity of what are called “Query Plans”. If you have never heard of a “Query Plan” you might want to skip this section as it will not be relevant, or understandable to you.

When a query is sent to any relational database the optimiser parses the query to determine what tables will be accessed. The optimiser then investigates the statistics that are held about each of the tables and the associated indexes and determines in what order, and using what mechanisms, the query will be performed.

Query optimiser technology is the “nuclear arms race” of relational databases. The optimiser is the “smarts” behind improving performance of queries for databases. They are extremely complicated and sophisticated today.

They have become very difficult to “second guess” and even more difficult to influence and make them do what you want them to do.

When using partitions the optimiser will report on which partitions will be accessed, what order they will be accessed in, and how they will be accessed, etc.

These Query Plans are really only understandable by experienced DBAs who are familiar with working with partitions.

The “average” DBAs who look after operational systems may never have seen a partitioned table in their career.

So even though database level partitioning may be presented as “the database does it all and you do not need to know anything about it” this is not really true.

When there are query performance issues the DBAs will need to read the Query Plans for the partitioned tables and they will need to understand them. If the DBAs have little experience in this area then it will take time to gain that experience.

By contrast, when using Manually Partitioned Tables implemented with MicroStrategy the queries are broken up in to separate sub queries each of which is easy to understand. In this way any DBA who has reasonable experience can read the query plans that are generated as they are much simpler.

Because the DBAs can more easily read the simpler query plans they are more able to positively influence performance. There is no need to have a DBA with experience in the SQL Server Partitioning Feature and the more complex query plans and no need to gain that experience.

Increased Reliability of SQL Server Performance Levels

When implementing Manually Partitioned Tables with MicroStrategy the queries that are being sent to the database are more easily optimised and resolved. The DBAs are more easily able to add indexes or tune queries because the way in which the query is resolved is simpler.

Because each component is simpler and more easily tuned there tends to be an increased reliability of performance levels when Manual Partitioning is used.

5. INTRODUCTION TO EXAMPLE PARTITION SALES TABLE

In this section we will introduce our example BIDA Sales Transaction Fact Table. In BIDA each fact table has a number and our Sales Transaction Fact Table is #532. This is why you will see the number 532 throughout this example.

In many companies the Sales Transaction Fact Table is a large table. Especially for companies where the items being sold are low cost per item. This happens in all sorts of retailers now that we have scanner data.

In almost every retailer of any significant size the Sales Transaction Fact Table should be partitioned so it makes an excellent example that most people can understand. If you work in a bank or a telco just think of the account transactions or call detail records, ok?

In our example we are going to create a very simple partitioning strategy. We are going to partition sales based on the month in which the item was sold. Even though this is a very simple partitioning strategy it is the #1 strategy used across the data warehousing world.

In our example we are going to create a fact table for each month worth of sales from January 2015 to December 2017. Of course most companies have sales transaction fact tables that are over a longer time period.

We will create 36 fact tables for these 36 months worth of sales and we will demonstrate how MicroStrategy generates SQL to query these 36 tables as if they are one table. The MicroStrategy user is not aware that s/he is accessing 36 tables unless they go and read the SQL.

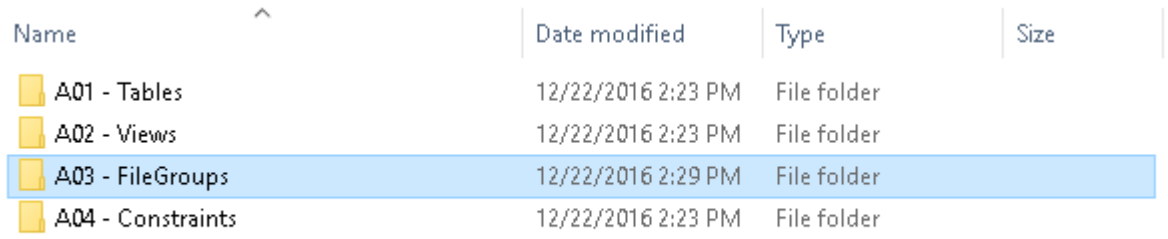
As mentioned this is for SQL Server Standard Edition 2014.

5.1. Creation of file groups

In SQL Server it is possible to place data from different partitions on to different disks that are visible from the SQL Server. This is done via two items called “File Groups” and “Files”.

It is obvious that a “File Group” can be a group of “Files”.

In our sample code, which you can request from us, you will see a directory called “A03 - FileGroups”.



Inside that directory you will find a file called FG0532 – File Groups.sql.

Inside that file you will find the following code to create the File Groups.

The first set of File Groups is for the actual data that will represent the sales for each month.

The naming standard should be obvious.

TF0532 = **T**able, **F**act, #**532** which happens to be the sales transaction fact table.

YYMM = the Year and month of the fact table data. Since we are in 2017 I rather think that we will not have to worry about the Y3K problem! We will be long retired (or dead) before that happens!

You can see that we add a File Group for each month of each year making 36 in total.

```
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1501;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1502;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1503;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1504;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1505;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1506;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1507;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1508;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1509;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1510;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1511;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1512;

ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1601;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1602;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1603;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1604;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1605;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1606;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1607;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1608;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1609;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1610;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1611;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1612;

ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1701;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1702;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1703;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1704;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1705;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1706;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1707;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1708;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1709;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1710;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1711;
ALTER   DATABASE  BIDA3000        ADD             FILEGROUP       FG_TF0532_1712;
```

The next set of file groups is for the indexes that we plan to create. Again we have created a file group for each month of each year and the indexes will go in to those file groups. We have used the suffix IX01 in case it is desired to create more file groups for more indexes at a later stage. We have made the naming standard robust enough for any reasonable sized company.

```
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1501_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1502_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1503_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1504_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1505_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1506_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1507_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1508_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1509_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1510_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1511_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1512_IX01;

ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1601_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1602_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1603_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1604_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1605_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1606_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1607_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1608_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1609_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1610_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1611_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1612_IX01;

ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1701_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1702_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1703_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1704_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1705_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1706_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1707_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1708_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1709_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1710_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1711_IX01;
ALTER DATABASE BIDA3000 ADD FILEGROUP FG_TF0532_1712_IX01;
```

In the file that you can obtain from us we have the remove commands as well. It does not seem sensible to include them in this document to discuss how to create the manually partitioned tables.

5.2. Creation of Files

The File Groups are logical objects merely used as definitions on how to direct data to the physical files that will make up the data warehouse. The File Groups do not actually create any files.

The next step of the process is the creation of the Files to allocate actual space on disk to hold data and linking those files to the File Groups.

You will find the following file in the sample code you can obtain from us.

You will find the following code inside that file. We will take one statement and explain it in detail and then present the following statements.

```
ALTER DATABASE BIDA3000 ADD FILE
(       NAME = FG_TF0532_1501, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK01\P_FG_TF0532_1501_a.ndf',
SIZE    = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB )
TO FILEGROUP FG_TF0532_1501 ;
```

To explain this statement we will address the components.

NAME = FG_TF0532_1501
This is the logical name of the file inside SQL Server. When we are only creating one file per file group we do not add a suffix number. Since it is possible to have many files inside a file group you may wish to suffix the 1501, YYMM, number.

FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK01\P_FG_TF0532_1501_a.ndf'
The FileName parameter tells SQL server where to create the physical file and what name to give it in the operating system.

Since this is an example for partitioning we are creating the files on the C drive of the demonstration machine. To give you an idea of how data could be distributed we have included a “DISKNN” level in the naming standard for these files. So, for example, you might introduce a naming standard that users a drive letter + database name to place data on disks.

An example would be **F:\BIDA3000\P_FG_TF0532_1501_a.ndf'**

This would be placing the January 2015 partition of data on to F drive for database BIDA3000. This is the sort of naming standard you might adopt with such a partitioning strategy.

SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB
As with any creation of a file in SQL Server you must tell the database the initial size of the file, the maximum size of the file, and by what increments the file will grow when the file becomes full. All SQL Server DBAs are familiar with these parameters.

TO FILEGROUP FG_TF0532_1501
This is the key statement to understand. In the prior file we created the FileGroups. These are just a logical entry for the database to know where to put data for a table. In this statement we are linking the physical file to the File Group. Obviously we could create many files in the file group. In this case we are only creating one file in a file group. The tables are placed in to a single FileGroup and that FileGroup then directs the data to be placed in to the files in the file group.

In this way you gain control of which disk drives, and in which directories and files, data for a table is sent to. Obviously it is possible to send data to different disk drives no matter how that drive is made visible to SQL Server.

In the following set of statements you can see that we are creating files for each month in 2015. I would like to draw your notice to the **DISK01**, **DISK02** etc that is in red. In order to demonstrate the way in which data can be distributed over different disk drives we have added the **DISKNN** level of the directory name for this example.

Depending on your volumes you may have 6 or more drives available to you to place data on. It is recommended that you place each months worth of data on to a different drive for as many drives as you have available that makes sense. If you have hundreds of drives it would only make sense to go back 12 months and cycle the use of drives across 12 months for different fact tables.

In this example we are cycling sales transactions over “drives” for a period of 6 months. This would mean, if implemented on six disks, that when scanning data from the sales transaction table there would be no contention on drives unless the period scanned is longer than 6 months. If a query was written to query data from 201501 to 201506 and that data was placed on 6 different drives then the reads would be from the 6 drives and would not create disk contention.

This placement of data across different drives, which may themselves be RAIDed at different levels, is very important to improving query performance as well as maintenance performance in the ETL system. The more the workload can be distributed across disk drives the faster the data warehouse can be made to work. And the differences can be very substantial.

When building operational systems many DBAs are trained by Microsoft to simply create one PRIMARY file group. They are taught to then link all the available disk to the primary file group and to place all data in to this one filegroup. Our colleagues at Oracle tend to train DBAs in a similar way. By doing this data for different tables are “mixed together” and so scanning such data takes much longer. Further, of a number of fact tables are being scanned at the same time the contention on the disk can be very significant.

Our experience has shown that setting up SQL Server databases like this can provide a very significant increase in query performance for the detailed fact tables as well as very significant ETL performance improvements.

```
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1501, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK01\P_FG_TF0532_1501_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1501 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1502, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK02\P_FG_TF0532_1502_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1502 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1503, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK03\P_FG_TF0532_1503_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1503 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1504, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK04\P_FG_TF0532_1504_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1504 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1505, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK05\P_FG_TF0532_1505_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1505 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1506, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK06\P_FG_TF0532_1506_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1506 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1507, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK01\P_FG_TF0532_1507_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1507 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1508, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK02\P_FG_TF0532_1508_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1508 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1509, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK03\P_FG_TF0532_1509_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1509 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1510, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK04\P_FG_TF0532_1510_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1510 ;
```

```
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1511, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK05\P_FG_TF0532_1511_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1511 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1512, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK06\P_FG_TF0532_1512_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1512 ;
```

Next we have the files being allocated for 2016.

```
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1601, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK01\P_FG_TF0532_1601_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1601 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1602, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK02\P_FG_TF0532_1602_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1602 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1603, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK03\P_FG_TF0532_1603_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1603 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1604, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK04\P_FG_TF0532_1604_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1604 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1605, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK05\P_FG_TF0532_1605_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1605 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1606, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK06\P_FG_TF0532_1606_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1606 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1607, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK01\P_FG_TF0532_1607_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1607 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1608, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK02\P_FG_TF0532_1608_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1608 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1609, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK03\P_FG_TF0532_1609_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1609 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1610, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK04\P_FG_TF0532_1610_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1610 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1611, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK05\P_FG_TF0532_1611_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1611 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1612, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK06\P_FG_TF0532_1612_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1612 ;
```

Next we have the files being allocated for 2017.

```
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1701, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK01\P_FG_TF0532_1701_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1701 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1702, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK02\P_FG_TF0532_1702_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1702 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1703, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK03\P_FG_TF0532_1703_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1703 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1704, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK04\P_FG_TF0532_1704_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1704 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1705, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK05\P_FG_TF0532_1705_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1705 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1706, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK06\P_FG_TF0532_1706_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1706 ;
```

```
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1707, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK01\P_FG_TF0532_1707_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1707 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1708, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK02\P_FG_TF0532_1708_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1708 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1709, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK03\P_FG_TF0532_1709_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1709 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1710, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK04\P_FG_TF0532_1710_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1710 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1711, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK05\P_FG_TF0532_1711_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1711 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1712, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK06\P_FG_TF0532_1712_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1712 ;
```

Next we have the files that we wish to place indexes in to. It is possible to place indexes in to different file groups than the tables thus gaining control over the indexes individually. In this example I wish to draw your attention to the DISKNN that is in red.

We recommend that the indexes that are placed on a months worth of data are actually placed on a different disk to the data itself. In this way when there are queries that will read both the index and the data the workload is spread across at least two disks. It also means that during updates the insertion of data and updates of indexes occurs on different disks. This also improves ETL performance.

```
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1501_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK02\P_FG_TF0532_1501_IX01_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1501_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1502_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK03\P_FG_TF0532_1502_IX01_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1502_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1503_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK04\P_FG_TF0532_1503_IX01_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1503_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1504_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK05\P_FG_TF0532_1504_IX01_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1504_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1505_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK06\P_FG_TF0532_1505_IX01_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1505_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1506_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK01\P_FG_TF0532_1506_IX01_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1506_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1507_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK02\P_FG_TF0532_1507_IX01_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1507_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1508_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK03\P_FG_TF0532_1508_IX01_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1508_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1509_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK04\P_FG_TF0532_1509_IX01_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1509_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1510_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK05\P_FG_TF0532_1510_IX01_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1510_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1511_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK06\P_FG_TF0532_1511_IX01_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1511_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE ( NAME = FG_TF0532_1512_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK01\P_FG_TF0532_1512_IX01_a.ndf', SIZE = 0001MB, MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1512_IX01 ;
```

The next set of statements are for the files in which to place indexes for 2016 and 2017.

```
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1601_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK02\P_FG_TF0532_1601_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1601_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1602_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK03\P_FG_TF0532_1602_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1602_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1603_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK04\P_FG_TF0532_1603_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1603_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1604_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK05\P_FG_TF0532_1604_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1604_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1605_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK06\P_FG_TF0532_1605_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1605_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1606_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK01\P_FG_TF0532_1606_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1606_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1607_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK02\P_FG_TF0532_1607_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1607_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1608_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK03\P_FG_TF0532_1608_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1608_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1609_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK04\P_FG_TF0532_1609_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1609_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1610_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK05\P_FG_TF0532_1610_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1610_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1611_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK06\P_FG_TF0532_1611_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1611_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1612_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK01\P_FG_TF0532_1612_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1612_IX01 ;
```

```
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1701_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK02\P_FG_TF0532_1701_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1701_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1702_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK03\P_FG_TF0532_1702_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1702_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1703_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK04\P_FG_TF0532_1703_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1703_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1704_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK05\P_FG_TF0532_1704_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1704_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1705_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK06\P_FG_TF0532_1705_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1705_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1706_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK01\P_FG_TF0532_1706_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1706_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1707_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK02\P_FG_TF0532_1707_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1707_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1708_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK03\P_FG_TF0532_1708_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1708_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1709_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK04\P_FG_TF0532_1709_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1709_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1710_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK05\P_FG_TF0532_1710_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1710_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1711_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK06\P_FG_TF0532_1711_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1711_IX01 ;
ALTER DATABASE BIDA3000 ADD FILE (	NAME = FG_TF0532_1712_IX01, FILENAME = 'C:\SQLSERVERDATA\PARTITIONS\DISK01\P_FG_TF0532_1712_IX01_a.ndf', SIZE	= 0001MB,
MAXSIZE = unlimited, FILEGROWTH = 0001MB ) TO FILEGROUP FG_TF0532_1712_IX01 ;
```

5.3. Creation of Tables

In the preceding sections we created the FileGroups and Files into which we will place data. There were no tables created. When creating Manually Partitioned Tables you need to think about where you are going to physically place the data to provide the best query and ETL update performance. The over-riding concern should be to maximise the throughput of the disk drives because the IO from the disks is still the slowest part of a query by a wide margin.

In the source code that is available from us on request you can find the following directory.

Name	Date modified	Type
A01 - Tables	12/22/2016 2:23 PM	File folder

In this directory you can file these files. One file per table. Each months worth of data will be stored in its own separate table.

Name	Date modified	Type	Size
tf_sale_txn_1501	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1502	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1503	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1504	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1505	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1506	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1507	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1508	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1509	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1510	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1511	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1512	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1601	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1602	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1603	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1604	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1605	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1606	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1607	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1608	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1609	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1610	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1611	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1612	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1701	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1702	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1703	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1704	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1705	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1706	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1707	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1708	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1709	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1710	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1711	12/21/2016 3:57 PM	TXT File	3 KB
tf_sale_txn_1712	12/21/2016 3:57 PM	TXT File	3 KB

Each file in the above list contains a create statement similar to the following.

CREATE TABLE [dbo].[tf_sale_txn_1501] – Notice the 1501. This is a YYMM suffix for each table.

```
([pk_tf_sale_txn]                    [int] NOT NULL,
[dk_vm_sale_date]                    [int] NOT NULL,
[dk_vm_sale_minute]                  [int] NOT NULL,
[dk_vm_sub_campaign]                 [int] NOT NULL,
[dk_vm_product]                      [int] NOT NULL,
[dk_vm_customer]                     [int] NOT NULL,
[dk_vf_customer_dims_asoc]           [int] NOT NULL,
[dk_vm_account]                      [int] NOT NULL,
[dk_vf_account_dims_asoc]            [int] NOT NULL,
[dk_vm_customer_demographic]         [int] NOT NULL,
[dk_vm_customer_geography]           [int] NOT NULL,
[dk_vm_geo_code]                     [int] NOT NULL,
[dk_vm_map_reference]                [int] NOT NULL,
[dk_vm_response_type]                [int] NOT NULL,
[dk_vm_currency]                     [int] NOT NULL,
[dk_vm_sales_rep]                    [int] NOT NULL,
[dk_vm_unit_of_measure]              [int] NOT NULL,
[dk_vm_txn_type]                     [int] NOT NULL,
[dk_vm_card]                         [int] NOT NULL,
[dk_vm_card_type]                    [int] NOT NULL,
[dk_vm_comp_reporting_struct]        [int] NOT NULL,
[dk_vm_account_status]               [int] NOT NULL,
[dk_vm_account_type]                 [int] NOT NULL,
[dk_vf_campaign_dims_asoc]           [int] NOT NULL,
[sale_unit_amount]                   [money] NULL,
[sale_extended_amount]               [money] NULL,
[cost_unit_amount]                   [money] NULL,
[cost_extended_amount]               [money] NULL,
[tax1_unit_amount]                   [money] NULL,
[tax1_extended_amount]               [money] NULL,
[tax2_unit_amount]                   [money] NULL,
[tax2_extended_amount]               [money] NULL,
[discount_unit_amount]               [money] NULL,
[discount_extended_amount]           [money] NULL,
[sale_units]                         [int] NULL,
[appsys_reference_number_str]        [nvarchar](2000) NULL,
[sale_sdesc]                         [nvarchar](2000) NULL,
[sale_ldesc]                         [nvarchar](2000) NULL,
[sale_tstamp]                        [datetime] NULL,
[dk_vf_solicitation_txn]             [int] NULL,
[number_sales]                       [int] NULL,
[sale_txn_returned_flag]             [char](1) NULL,
[return_period_expired_flag]         [char](1) NULL,
[return_probability_pct]             [decimal](38, 10) NULL,
[table_number]                       [int] NOT NULL,
[row_del_from_src_ind]               [char](1) NULL,
[partitioning_column]                [int] NOT NULL,
[partitioning_column_part_01]        [int] NULL,
[partitioning_column_part_02]        [int] NULL,
[batch_number]                       [int] NOT NULL,
[file_cycle_number]                  [int] NOT NULL,
[view_number]                        [int] NOT NULL,
[audit_timestamp_01]                 [datetime] NULL,
[ss_number]                          [int] NOT NULL,
[field_setter]                       [int] NULL
```

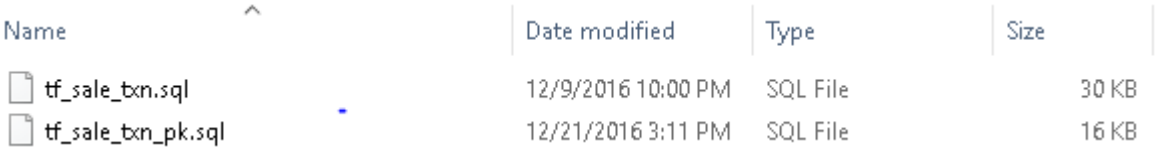
) ON [FG_TF0532_1501] – This is where the file group is named to link the table to the physical files where the data will be inserted. Each table is placed on its own file group.

5.4. Creation of Constraints

With Manually Partitioned Tables as we are creating in our example it is necessary to create a primary key on the table to be able to perform ETL on the collection of tables joined together in a Union View. This is not part of the purpose of this demonstration, however, to make it clear how to properly create the tables we have included the creation of the constraints and Primary Keys for completeness.

In the code that you can obtain from us you will find the following folder.

It will contain the following files.



The format to create the constraints for the Primary Key for each table is as follows.

The most important components of the statement are marked in red and will be explained after the statement.

```
ALTER TABLE [dbo].[tf_sale_txn_1501] ADD  CONSTRAINT [pk_tf_sale_txn_1501] PRIMARY KEY NONCLUSTERED
(
        [partitioning_column] ASC
       ,[pk_tf_sale_txn] ASC
)WITH (PAD_INDEX = ON, STATISTICS_NORECOMPUTE = ON, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF,
ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, FILLFACTOR = 85) ON [FG_TF0532_1501_IX01]
```

- tf_sale_txn_1501 – This is the name of the table on which the constraint is to be added. The naming standard is to suffix the table with YYMM.
- pk_tf_sale_txn_1501 – This is the name of the primary key constraint that will be applied to the table.
- NONCLUSTERED – We have found that table management of very large tables, as well as query performance, is better with non clustered tables.
- FG_TF0532_1501_IX01 – This is the clause that links the index to the File Group in which the index will be placed. This in turn tells SQL Server where to place the data on the disks that are available to SQL Server.

Of course there is one of these alter table statements for each separate physical table that will make up part of the Manually Partitioned Fact Table.

To enable the ETL to properly place the data in to the Manual Partitions each table must have a constraint defined for a partitioning column that will be set for each row. This constraint can also be a range such as between two values. The between two values would be used if the date of a transaction is available but the month id of the row is not available.

In BIDA we have defined a field called partitioning_column. This field is an integer which will place the row in one of the partitions that has been defined.

The statement is very simple and the syntax is obvious.

The other requirement of SQL Server is that the partitioning column must be available in the primary key and that each manual partition must have a primary key.

In BIDA we have a standard that every fact record gets a primary key added to the front of the record. This is an integer except when it is expected that more than 2 billion rows will be eventually placed in to the partitioned fact table. Then we use big integers for the primary key.

It is true that the primary key is sufficient without the partitioning field being added to it. However SQL Server insists that the field that is used to partition the tables is present in both the primary key definition constraint of the table and in a separate constraint that the database can see.

Again, this constraint is required for the ETL to work through views and is not required for the query processing using MicroStrategy.

```
ALTER TABLE dbo.tf_sale_txn_1501  ADD CONSTRAINT chk_tf_sale_txn_1501_01 CHECK (partitioning_column = 201501 );
ALTER TABLE dbo.tf_sale_txn_1502  ADD CONSTRAINT chk_tf_sale_txn_1502_01 CHECK (partitioning_column = 201502 );
ALTER TABLE dbo.tf_sale_txn_1503  ADD CONSTRAINT chk_tf_sale_txn_1503_01 CHECK (partitioning_column = 201503 );
ALTER TABLE dbo.tf_sale_txn_1504  ADD CONSTRAINT chk_tf_sale_txn_1504_01 CHECK (partitioning_column = 201504 );
ALTER TABLE dbo.tf_sale_txn_1505  ADD CONSTRAINT chk_tf_sale_txn_1505_01 CHECK (partitioning_column = 201505 );
ALTER TABLE dbo.tf_sale_txn_1506  ADD CONSTRAINT chk_tf_sale_txn_1506_01 CHECK (partitioning_column = 201506 );
ALTER TABLE dbo.tf_sale_txn_1507  ADD CONSTRAINT chk_tf_sale_txn_1507_01 CHECK (partitioning_column = 201507 );
ALTER TABLE dbo.tf_sale_txn_1508  ADD CONSTRAINT chk_tf_sale_txn_1508_01 CHECK (partitioning_column = 201508 );
ALTER TABLE dbo.tf_sale_txn_1509  ADD CONSTRAINT chk_tf_sale_txn_1509_01 CHECK (partitioning_column = 201509 );
ALTER TABLE dbo.tf_sale_txn_1510  ADD CONSTRAINT chk_tf_sale_txn_1510_01 CHECK (partitioning_column = 201510 );
ALTER TABLE dbo.tf_sale_txn_1511  ADD CONSTRAINT chk_tf_sale_txn_1511_01 CHECK (partitioning_column = 201511 );
ALTER TABLE dbo.tf_sale_txn_1512  ADD CONSTRAINT chk_tf_sale_txn_1512_01 CHECK (partitioning_column = 201512 );
```

```
ALTER TABLE dbo.tf_sale_txn_1601  ADD CONSTRAINT chk_tf_sale_txn_1601_01 CHECK (partitioning_column = 201601 );
ALTER TABLE dbo.tf_sale_txn_1602  ADD CONSTRAINT chk_tf_sale_txn_1602_01 CHECK (partitioning_column = 201602 );
ALTER TABLE dbo.tf_sale_txn_1603  ADD CONSTRAINT chk_tf_sale_txn_1603_01 CHECK (partitioning_column = 201603 );
ALTER TABLE dbo.tf_sale_txn_1604  ADD CONSTRAINT chk_tf_sale_txn_1604_01 CHECK (partitioning_column = 201604 );
ALTER TABLE dbo.tf_sale_txn_1605  ADD CONSTRAINT chk_tf_sale_txn_1605_01 CHECK (partitioning_column = 201605 );
ALTER TABLE dbo.tf_sale_txn_1606  ADD CONSTRAINT chk_tf_sale_txn_1606_01 CHECK (partitioning_column = 201606 );
ALTER TABLE dbo.tf_sale_txn_1607  ADD CONSTRAINT chk_tf_sale_txn_1607_01 CHECK (partitioning_column = 201607 );
ALTER TABLE dbo.tf_sale_txn_1608  ADD CONSTRAINT chk_tf_sale_txn_1608_01 CHECK (partitioning_column = 201608 );
ALTER TABLE dbo.tf_sale_txn_1609  ADD CONSTRAINT chk_tf_sale_txn_1609_01 CHECK (partitioning_column = 201609 );
ALTER TABLE dbo.tf_sale_txn_1610  ADD CONSTRAINT chk_tf_sale_txn_1610_01 CHECK (partitioning_column = 201610 );
ALTER TABLE dbo.tf_sale_txn_1611  ADD CONSTRAINT chk_tf_sale_txn_1611_01 CHECK (partitioning_column = 201611 );
ALTER TABLE dbo.tf_sale_txn_1612  ADD CONSTRAINT chk_tf_sale_txn_1612_01 CHECK (partitioning_column = 201612 );

ALTER TABLE dbo.tf_sale_txn_1701  ADD CONSTRAINT chk_tf_sale_txn_1701_01 CHECK (partitioning_column = 201701 );
ALTER TABLE dbo.tf_sale_txn_1702  ADD CONSTRAINT chk_tf_sale_txn_1702_01 CHECK (partitioning_column = 201702 );
ALTER TABLE dbo.tf_sale_txn_1703  ADD CONSTRAINT chk_tf_sale_txn_1703_01 CHECK (partitioning_column = 201703 );
ALTER TABLE dbo.tf_sale_txn_1704  ADD CONSTRAINT chk_tf_sale_txn_1704_01 CHECK (partitioning_column = 201704 );
ALTER TABLE dbo.tf_sale_txn_1705  ADD CONSTRAINT chk_tf_sale_txn_1705_01 CHECK (partitioning_column = 201705 );
ALTER TABLE dbo.tf_sale_txn_1706  ADD CONSTRAINT chk_tf_sale_txn_1706_01 CHECK (partitioning_column = 201706 );
ALTER TABLE dbo.tf_sale_txn_1707  ADD CONSTRAINT chk_tf_sale_txn_1707_01 CHECK (partitioning_column = 201707 );
ALTER TABLE dbo.tf_sale_txn_1708  ADD CONSTRAINT chk_tf_sale_txn_1708_01 CHECK (partitioning_column = 201708 );
ALTER TABLE dbo.tf_sale_txn_1709  ADD CONSTRAINT chk_tf_sale_txn_1709_01 CHECK (partitioning_column = 201709 );
ALTER TABLE dbo.tf_sale_txn_1710  ADD CONSTRAINT chk_tf_sale_txn_1710_01 CHECK (partitioning_column = 201710 );
ALTER TABLE dbo.tf_sale_txn_1711  ADD CONSTRAINT chk_tf_sale_txn_1711_01 CHECK (partitioning_column = 201711 );
ALTER TABLE dbo.tf_sale_txn_1712  ADD CONSTRAINT chk_tf_sale_txn_1712_01 CHECK (partitioning_column = 201712 );
```

5.5. Creation of Views

At this point we have created all the physical tables, the primary key index, the needed files and file groups, and the constraints for the tables. These are all objects that the DBAs create.

In BIDA we have a standard that says that no access will be performed against the database via a physical table name. We insist that all access to the database for query tools is performed via views. This gives the DBAs much more freedom in managing physical tables without impacting the query applications. It also provides the ability to implement table naming standards without changing the naming standards of the views in BIDA.

Many companies have table naming standards that are different to BIDA. So we allow customers to implement their own table naming standards while maintaining the view names of BIDA.

In the source code available from us you will find the following folder.

In this folder you will find the following files.

Name	Date modified	Type	Size
vf_sale_txn_1501.sql	12/21/2016 3:36 PM	SQL File	2 KB
vf_sale_txn_1502.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1503.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1504.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1505.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1506.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1507.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1508.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1509.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1510.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1511.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1512.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1601.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1602.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1603.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1604.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1605.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1606.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1607.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1608.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1609.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1610.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1611.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1612.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1701.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1702.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1703.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1704.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1705.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1706.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1707.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1708.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1709.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1710.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1711.sql	12/21/2016 3:45 PM	SQL File	2 KB
vf_sale_txn_1712.sql	12/21/2016 3:45 PM	SQL File	2 KB

Each of the views is named for an individual table. In our example the only field that is renamed is the primary key field. However, in your implementation you might have used a different naming standard for the underlying tables.

```
drop view [dbo].[vf_sale_txn_1501]
go

CREATE view [dbo].[vf_sale_txn_1501]
as select
        [pk_tf_sale_txn]        pk_vf_sale_txn,
…
from tf_sale_txn_1501
```

There are 36 such views created, one over the top of each table.

6. EXAMPLE: MICROSTRATEGY IMPLEMENTATION.

The prior sections have documented how to create the separate physical tables that will contain the Manual Partitions so that you can use the lower levels of SQL Server rather than the Enterprise Edition Version of SQL Server.

In this section we will document what is necessary so that MicroStrategy can communicate to the database in such a way as to only query those tables that are necessary to answer any particular question.

6.1. Partitioning Schema

For fact tables we have created a table that records the Partitioning Schema for fact tables. The SQL to create this table is as follows. Note that we have used the Primary FileGroup simply because this is a demonstration.

```
CREATE TABLE [dbo].[bida_mstr_partitions](
	[pk_int_key]                    [int] NOT NULL,
	[fact_table_name]               [varchar](255) NOT NULL,
	[partition_integer]             [int] NOT NULL,
	[PBTName]                       [varchar](255) NOT NULL,

	[partitioning_column_part_01]      [int] NOT NULL CONSTRAINT
	[DF_bida_mstr_partitions_partitioning_column_part_01]  DEFAULT ((0)),

	[partitioning_column_part_02] [int] NOT NULL CONSTRAINT
	[DF_bida_mstr_partitions_partitioning_column_part_02]  DEFAULT ((0)),

	[partitioning_column] [int] NOT NULL CONSTRAINT
	[DF_bida_mstr_partitions_partitioning_column]  DEFAULT ((0)),

 CONSTRAINT [pk_bida_mstr_partitions] PRIMARY KEY CLUSTERED
(
	[pk_int_key] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, FILLFACTOR = 80) ON [PRIMARY]
) ON [PRIMARY]
```

The data that is in this table for this example is as follows.
Note the last few rows are not captured in the screen capture.

pk_int_key	fact_table_name	partition_integer	PBTName	partitio...	partitio...	partitioning_column
1	vf_sale_txn	201501	vf_sale_txn_1501	0	0	0
2	vf_sale_txn	201502	vf_sale_txn_1502	0	0	0
3	vf_sale_txn	201503	vf_sale_txn_1503	0	0	0
4	vf_sale_txn	201504	vf_sale_txn_1504	0	0	0
5	vf_sale_txn	201505	vf_sale_txn_1505	0	0	0
6	vf_sale_txn	201506	vf_sale_txn_1506	0	0	0
7	vf_sale_txn	201507	vf_sale_txn_1507	0	0	0
8	vf_sale_txn	201508	vf_sale_txn_1508	0	0	0
9	vf_sale_txn	201509	vf_sale_txn_1509	0	0	0
10	vf_sale_txn	201510	vf_sale_txn_1510	0	0	0
11	vf_sale_txn	201511	vf_sale_txn_1511	0	0	0
12	vf_sale_txn	201512	vf_sale_txn_1512	0	0	0
13	vf_sale_txn	201601	vf_sale_txn_1601	0	0	0
14	vf_sale_txn	201602	vf_sale_txn_1602	0	0	0
15	vf_sale_txn	201603	vf_sale_txn_1603	0	0	0
16	vf_sale_txn	201604	vf_sale_txn_1604	0	0	0
17	vf_sale_txn	201605	vf_sale_txn_1605	0	0	0
18	vf_sale_txn	201606	vf_sale_txn_1606	0	0	0
19	vf_sale_txn	201607	vf_sale_txn_1607	0	0	0
20	vf_sale_txn	201608	vf_sale_txn_1608	0	0	0
21	vf_sale_txn	201609	vf_sale_txn_1609	0	0	0
22	vf_sale_txn	201610	vf_sale_txn_1610	0	0	0
23	vf_sale_txn	201611	vf_sale_txn_1611	0	0	0
24	vf_sale_txn	201612	vf_sale_txn_1612	0	0	0
25	vf_sale_txn	201701	vf_sale_txn_1701	0	0	0
26	vf_sale_txn	201702	vf_sale_txn_1702	0	0	0
27	vf_sale_txn	201703	vf_sale_txn_1703	0	0	0
28	vf_sale_txn	201704	vf_sale_txn_1704	0	0	0
29	vf_sale_txn	201705	vf_sale_txn_1705	0	0	0

This table is used to record the partition information for all fact tables.

Fact tables are usually partitioned by month. However, by having this underlying table it is possible to partition fact tables by any integer that you wish to partition by.

In order for MicroStrategy to communicate to the database via a Partitioning Schema a separate view needs to be created which exposes the views just for that fact table.

For the sales transaction fact table that view is defined as follows.

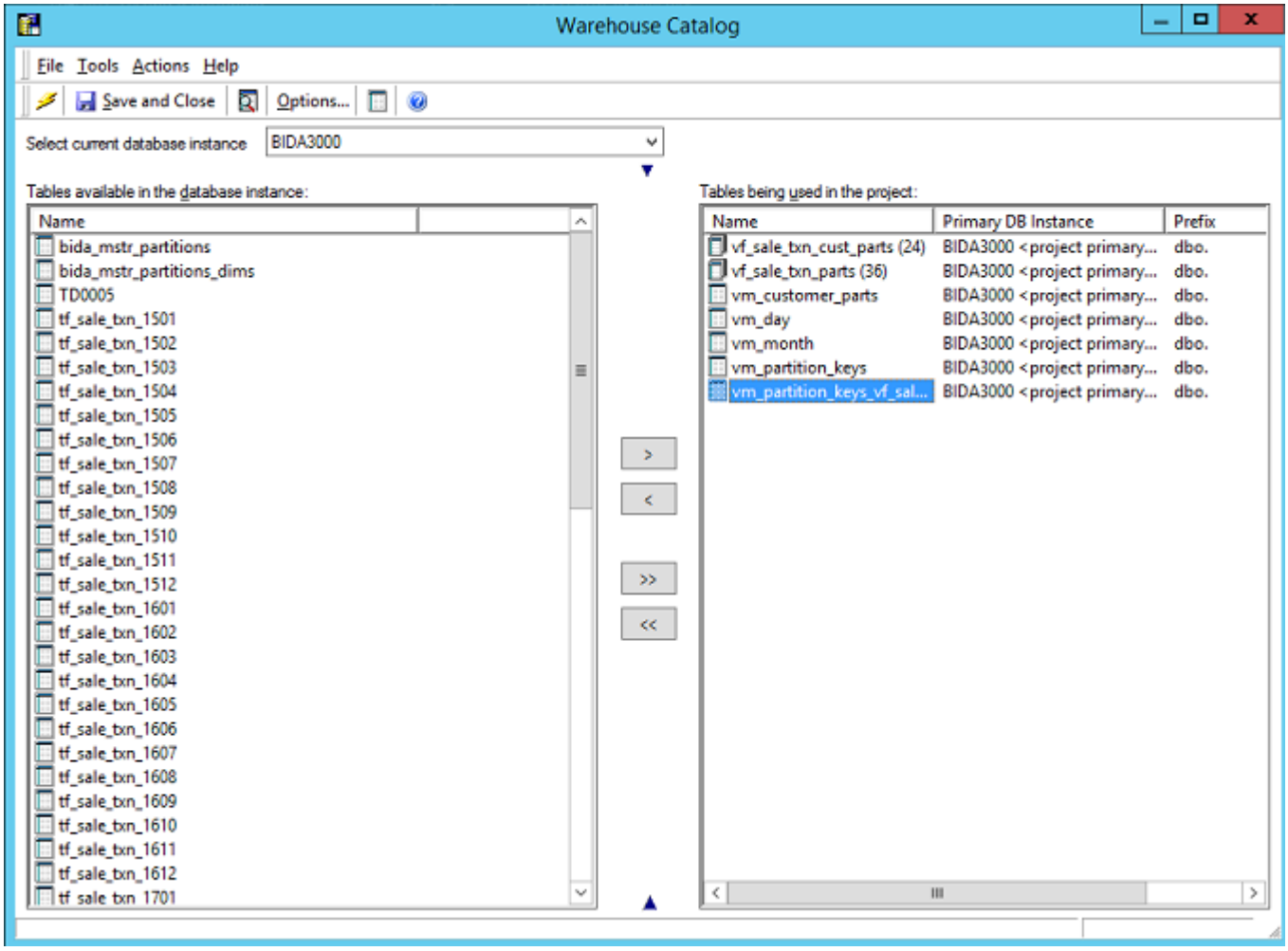
```
create view [dbo].[vf_sale_txn_parts]
as
select

        [partition_integer] [partitioning_column] ,
        [PBTName]

        from [dbo].[bida_mstr_partitions]

        where [fact_table_name] = 'vf_sale_txn'
```

The data from inside this view being presented to MicroStrategy is as follows.

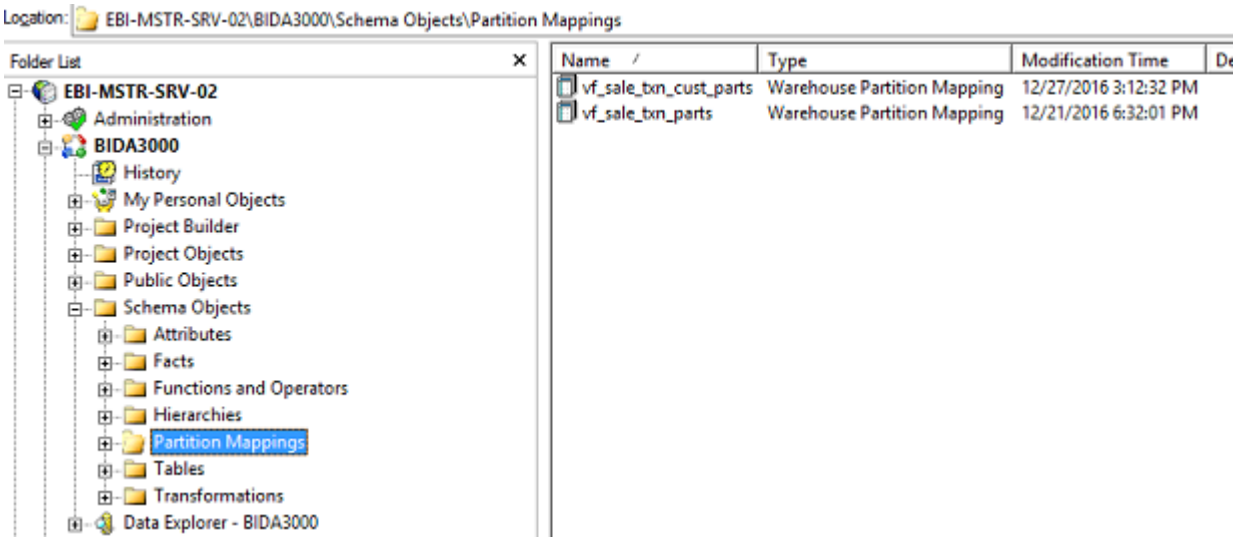
partitioning_column	PBTName
201501	vf_sale_txn_1501
201502	vf_sale_txn_1502
201503	vf_sale_txn_1503
201504	vf_sale_txn_1504
201505	vf_sale_txn_1505
201506	vf_sale_txn_1506
201507	vf_sale_txn_1507
201508	vf_sale_txn_1508
201509	vf_sale_txn_1509
201510	vf_sale_txn_1510
201511	vf_sale_txn_1511
201512	vf_sale_txn_1512
201601	vf_sale_txn_1601
201602	vf_sale_txn_1602
201603	vf_sale_txn_1603
201604	vf_sale_txn_1604
201605	vf_sale_txn_1605
201606	vf_sale_txn_1606
201607	vf_sale_txn_1607
201608	vf_sale_txn_1608
201609	vf_sale_txn_1609
201610	vf_sale_txn_1610
201611	vf_sale_txn_1611
201612	vf_sale_txn_1612
201701	vf_sale_txn_1701
201702	vf_sale_txn_1702
201703	vf_sale_txn_1703
201704	vf_sale_txn_1704
201705	vf_sale_txn_1705
201706	vf_sale_txn_1706

To link MicroStrategy to this partitioning schema you import the above view in to the warehouse catalog. This is done as follows.

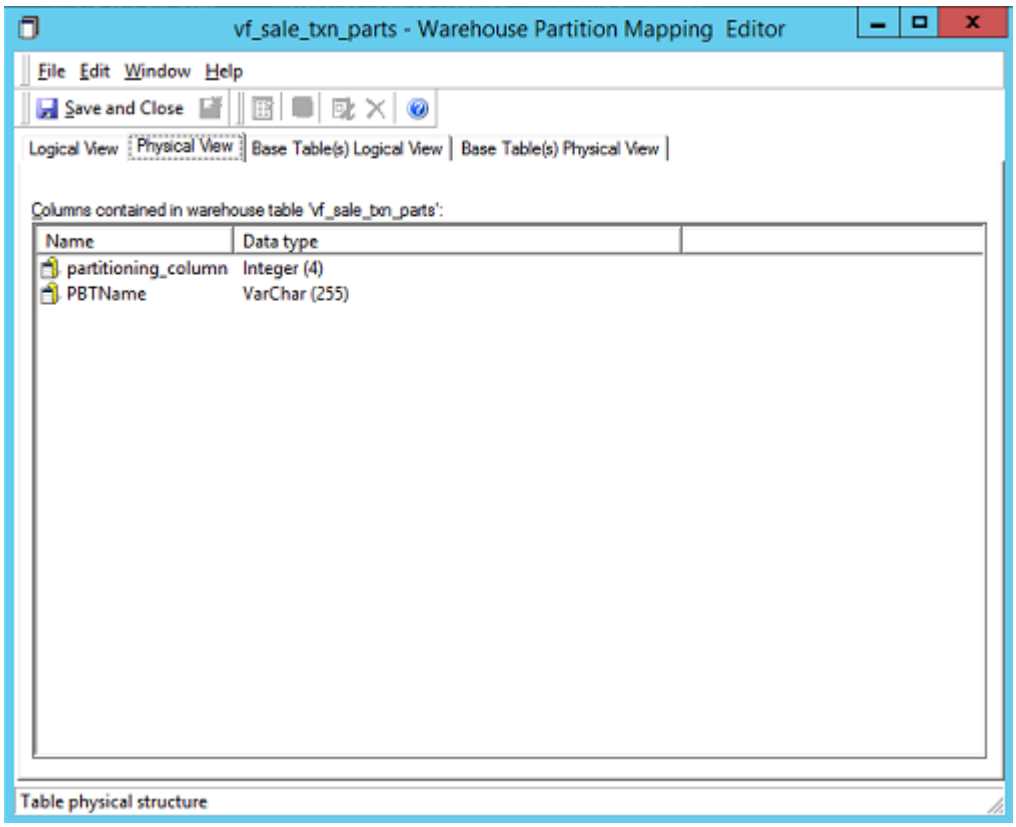
In the screen shot below you can see that vf_sale_txn_parts has 36 elements. These are the 3 years of tables for 2015, 16, 17. The Warehouse Catalog will read the data in the view and realise that you are wishing to define a Manually Partitioned Table and create the 36 elements for you.



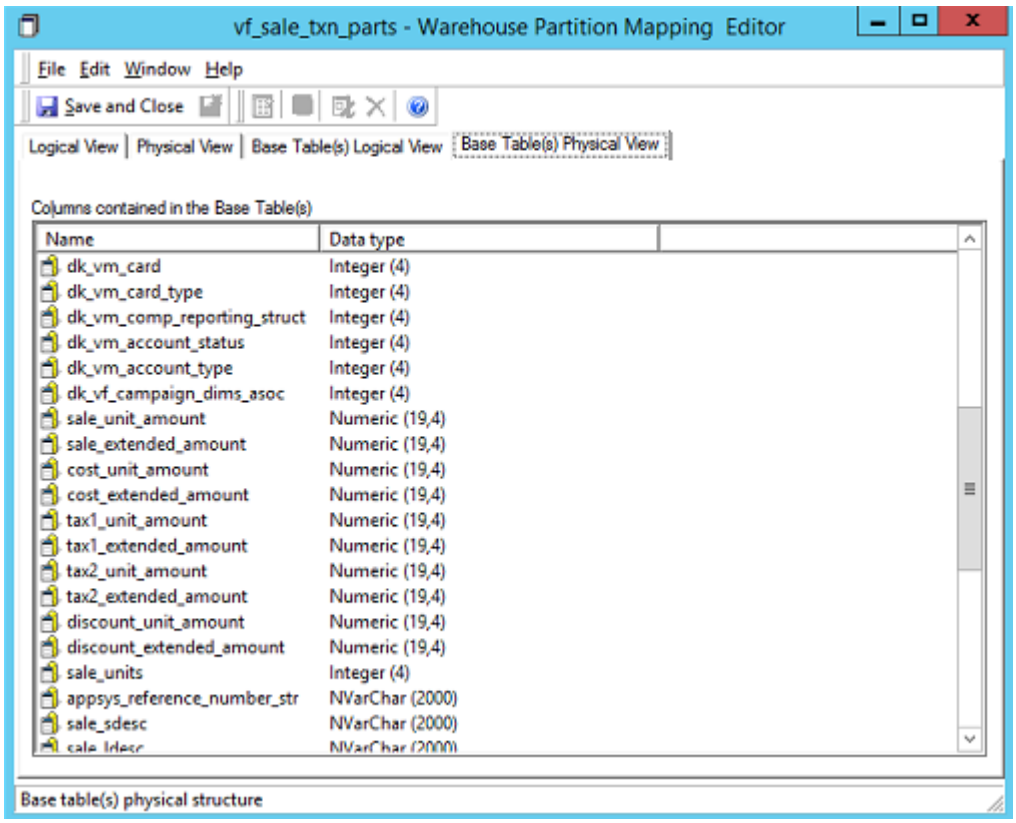
To show how this works you can navigate to Partition Mappings and you will see something similar to the following.



MicroStrategy will show the physical view of the table as follows.



And the Base Tables Physical View as follows:



So it is clear to see that the partitions have been read in and the fields are available for query even though they are in partitions.

6.2. Attribute definitions

The next portion of the set up is to create the attributes that will be used to inform MicroStrategy how to generate SQL to access the Manual Partitions.

To do this we need to create a number of Attributes.

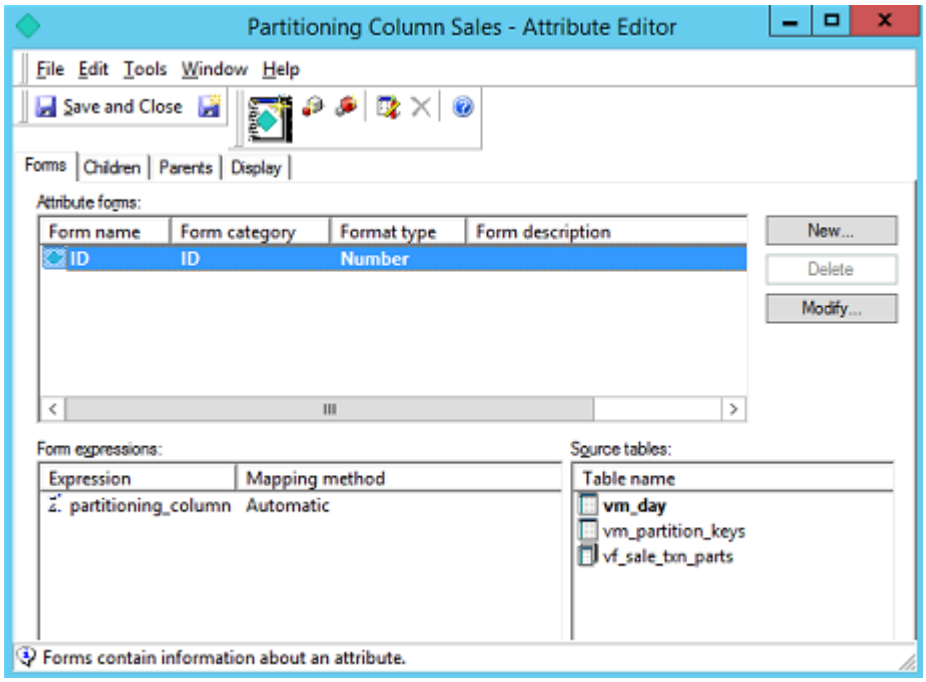
Name	Type	Modification Time
Partitioning Column Sales	Attribute	12/27/2016 12:24:53 PM
Partitioning Column Sales Cust	Attribute	12/27/2016 12:26:43 PM
Pk Vm Customer Parts	Attribute	12/27/2016 12:58:34 PM
Pk Vm Day	Attribute	12/27/2016 2:47:09 PM
Pk Vm Month	Attribute	12/27/2016 3:12:32 PM

In this example you can ignore Partitioning Column Sales Cust and Pk Vm Customer Parts as they are for the example where we are partitioning sales data by customers as well.

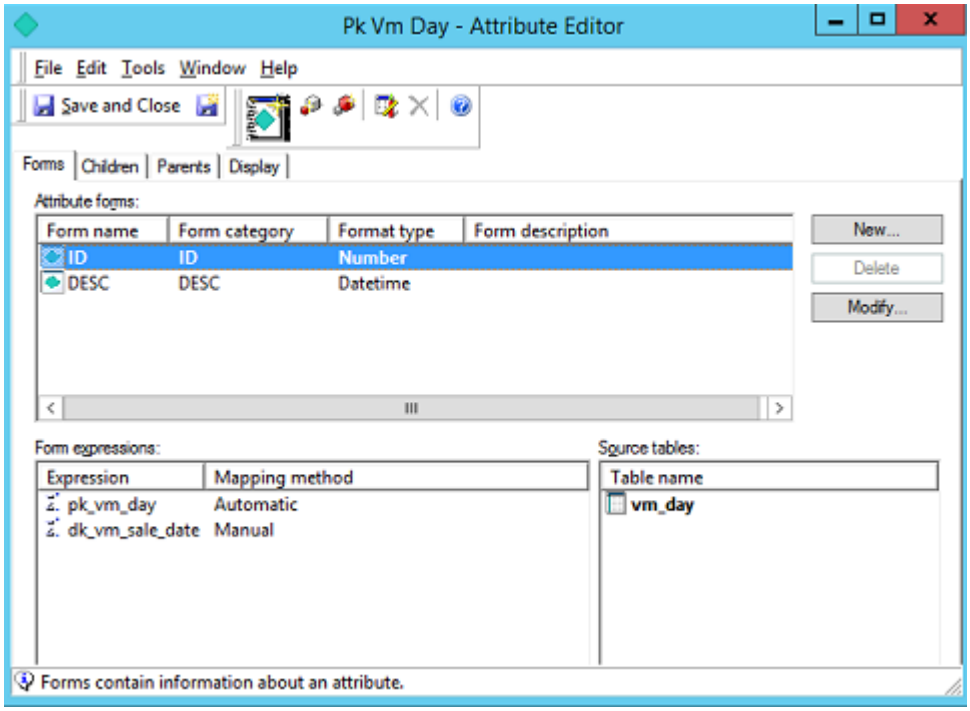
The attributes created for our example are Partitioning Column Sales, Pk Vm Day and Pk Vm Month.

Of course there would normally be many more attributes, this is just an example.

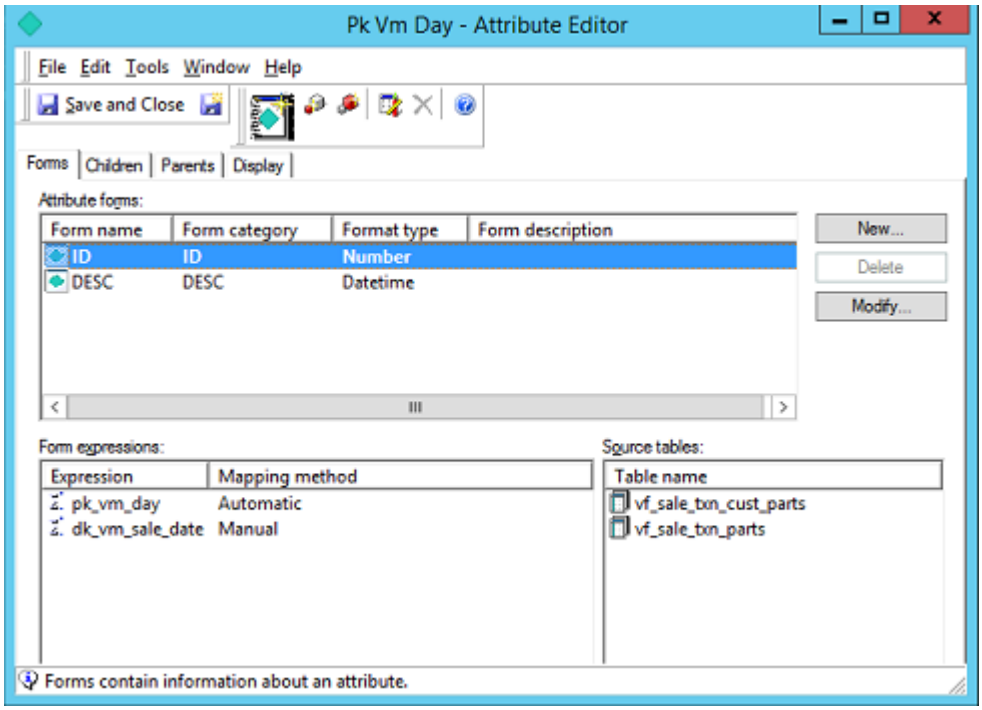
The attribute Partitioning Column Sales can be found on the tables vm_day, vm_partitioning_keys and vf_sale_txn_parts. Essentially it is the MicroStrategy month id that you will be familiar with.



Obviously Pk Vm Day is made visible from the vm_day table as follows.



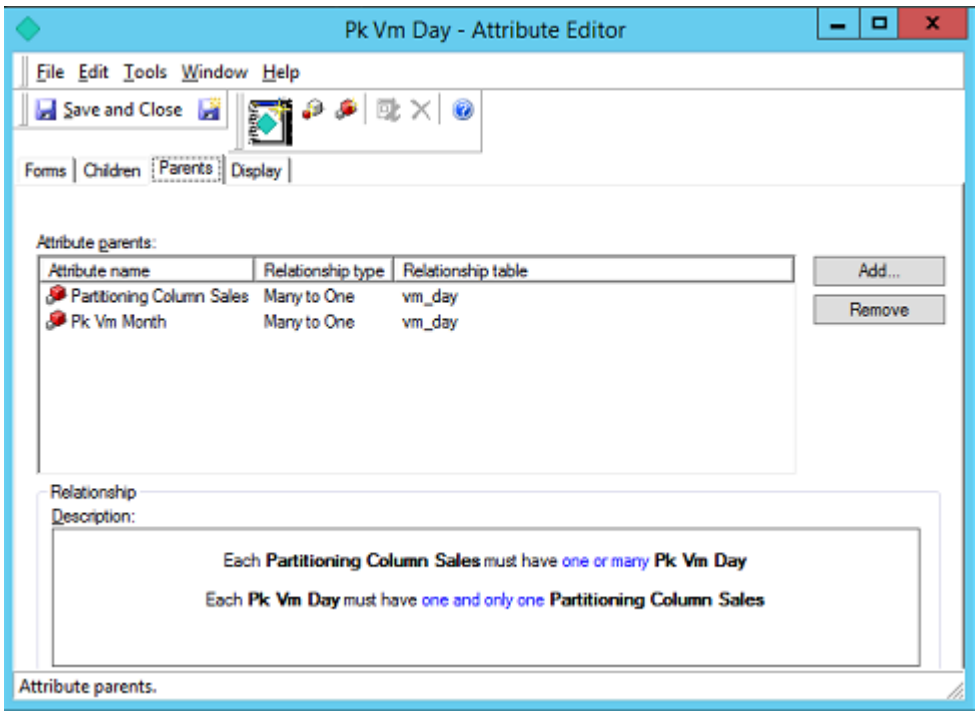
However, it is linked to the dk_vm_sale_date on the vf_sale_txn_parts table which is really the set of 36 views which make up Manual Partitions for 3 years. You can ignore cust_parts as it is part of another demonstration.



It is important to notice that when you are going to partition data in MicroStrategy by day and the detailed table does not have the month ID on it then you must place the partitioning key on the vm_day level table so that it will be available to MicroStrategy during the SQL generation processing. This is why the partitioning column must also be on the vm_day table (or similar) for partitioned fact tables that are partitioned over time.

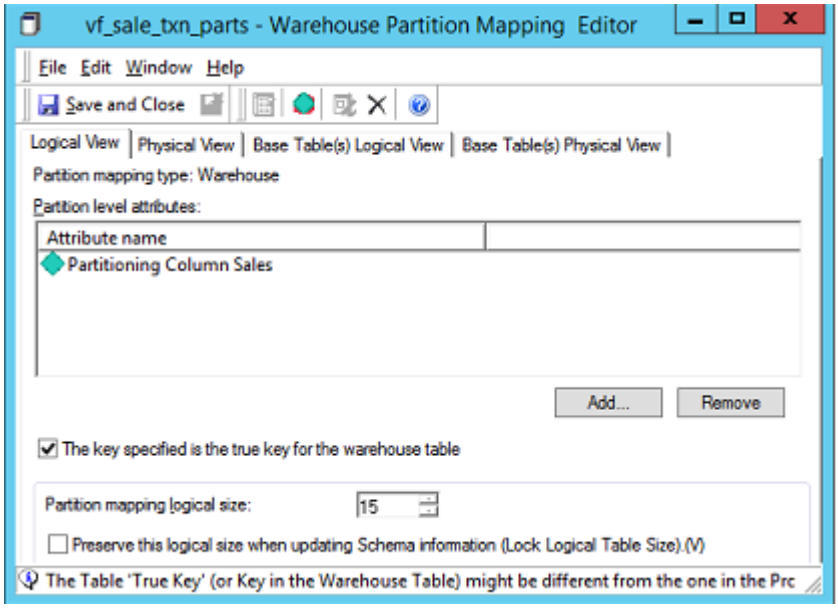
Since the partitioning column will be at the month level and the day table is at the day level it is necessary to define the partitioning column for sales as the parent for the pk vm day attribute as follows.

Of course, the month is also defined as a parent as follows as well.



There is now one last thing to do to set up MicroStrategy so that it can properly generate the SQL to read these partitions. This is to link the Partition Level Attribute to the Partition Mapping as follows.

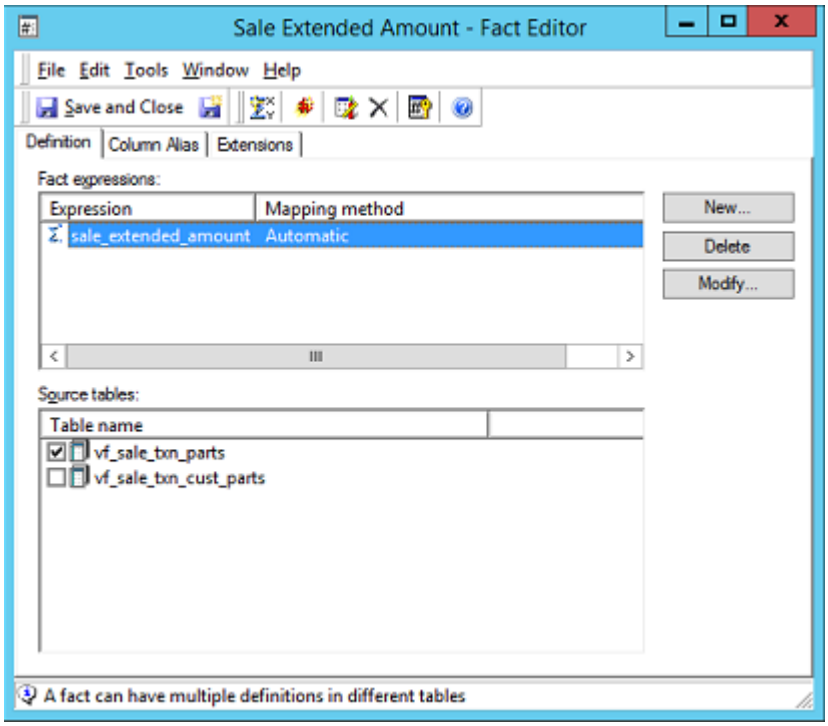
You can see in the Partition Mapping Editor we have added the Partition Column Sales attribute as the Partition Level Attributes.



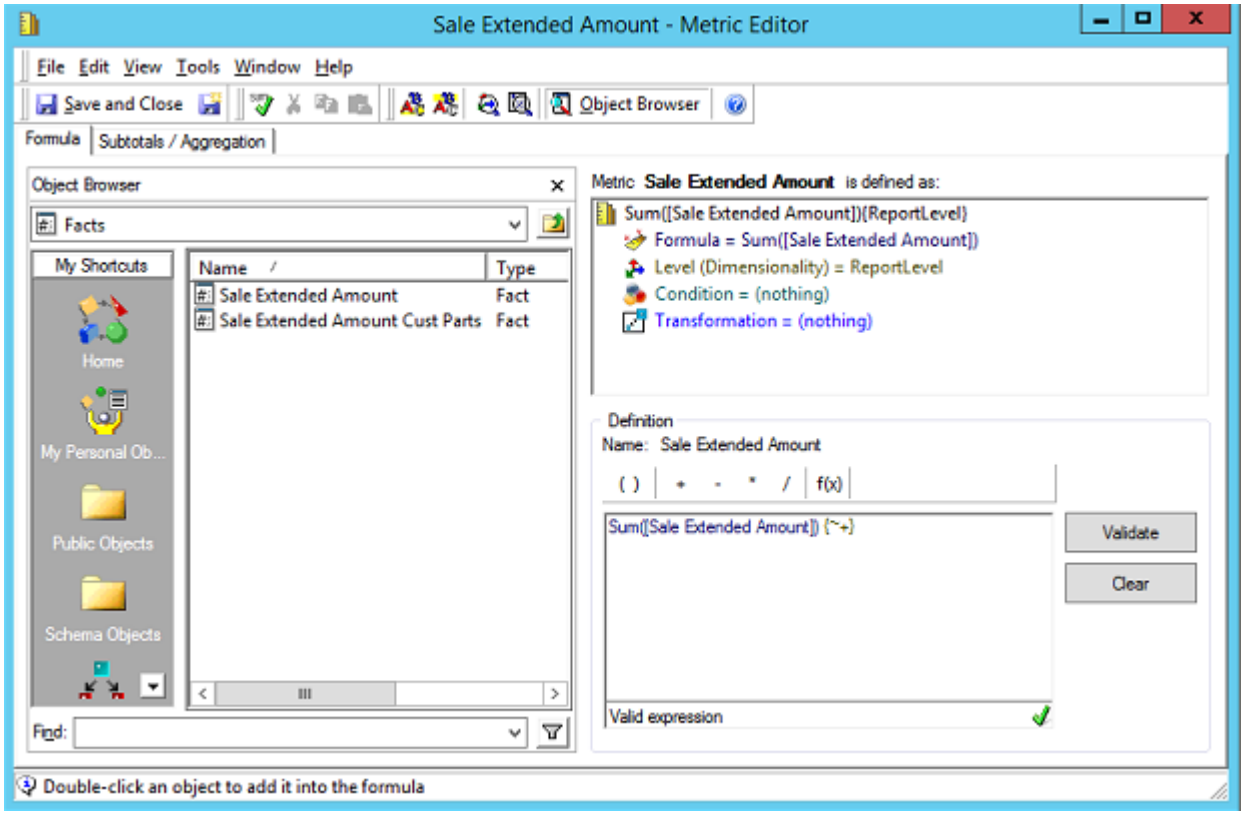
6.3. Fact And Metric Definitions

In order to write our first report we must create a fact and a metric to place on to the report. In our case we will create a fact and a metric for “extended sales amount” which is the gross sales amount on the sales fact table. This is simple. All MicroStrategy designers know how to do this so we will just show the screen shots of the results.

The first screen shot shows the sales_extended_amount coming from the vf_sales_txn_parts table.



This second screen shot show the Sales Extended Amount metric coming from the Sales Extended Amount fact. This is all very normal for schema designers.

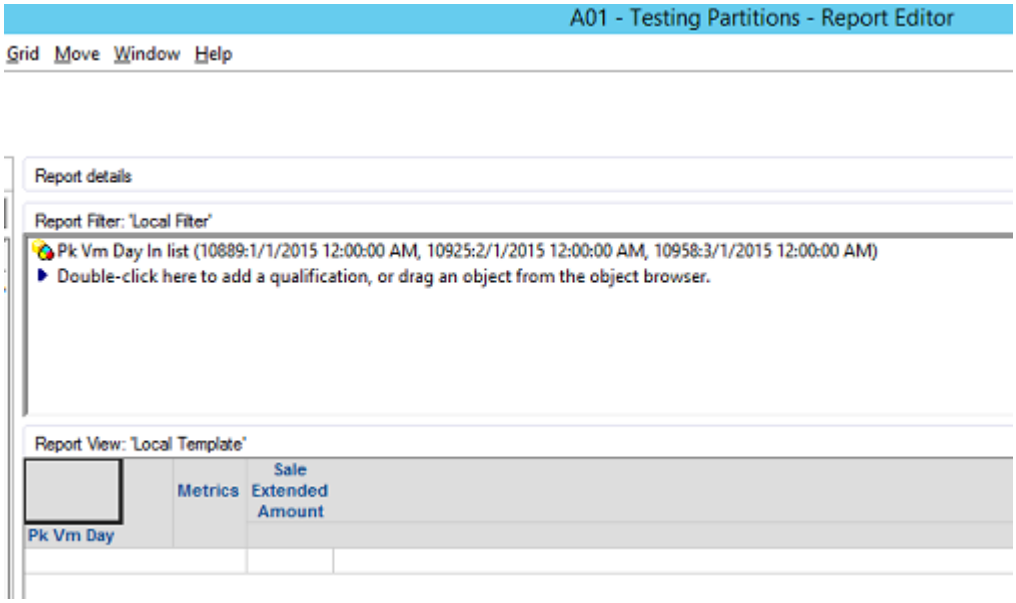


6.4. Example Report For Manually Partitioned Tables

Now we are now ready to write our first Report to generate a query that will properly access the correct partitions based on the data that is selected.

So now we create a very simple report to select sales data for 3 days that occur in different months. The report is as follows. You can see we have added filters for the 1st of January, February and March in 2015.

We are asking for the day and the Sale Extended Amount.



6.5. Example SQL

MicroStrategy will now use all the information we have provided it to generate highly efficient SQL that only reads the tables that are necessary to be read. It does this in a number of steps.

```
Pass0 -          Query Pass Start Time:          12/29/2016 3:45:56 PM
         Query Pass End Time:            12/29/2016 3:45:56 PM
         Query Execution:        0:00:00.00
         Data Fetching and Processing:  0:00:00.00
           Data Transfer from Datasource(s):      0:00:00.00
         Other Processing:       0:00:00.04
         Rows selected: 3

select   distinct a11.PBTNAME  PBTNAME
from     dbo.vf_sale_txn_parts   a11
         join      dbo.vm_day      a12
           on      (a11.partitioning_column = a12.partitioning_column)
where    a12.pk_vm_day in (10889, 10925, 10958)
```

Notice that in this step it joins the vm_day table with the vf_sale_txn_parts table to retrieve the PBTNAME where the pk_vm_day is in the set of key that define the three days selected. This will return the names of the tables in which the data that can possibly satisfy the query will be stored.

You can see that “Rows Selected” is 3 which is what you would expect.

The SQL then generated is as follows.

```
select    a11.dk_vm_sale_date  pk_vm_day,
          max(a12.day_date)  day_date,
          sum(a11.sale_extended_amount)  WJXBFS1
from      dbo.vf_sale_txn_1501        a11
          join      dbo.vm_day        a12
            on      (a11.dk_vm_sale_date = a12.pk_vm_day)
where     a11.dk_vm_sale_date in (10889, 10925, 10958)
group by a11.dk_vm_sale_date

union all

select    a11.dk_vm_sale_date  pk_vm_day,
          max(a12.day_date)  day_date,
          sum(a11.sale_extended_amount)  WJXBFS1
from      dbo.vf_sale_txn_1502        a11
          join      dbo.vm_day        a12
            on      (a11.dk_vm_sale_date = a12.pk_vm_day)
where     a11.dk_vm_sale_date in (10889, 10925, 10958)
group by a11.dk_vm_sale_date

union all

select    a11.dk_vm_sale_date  pk_vm_day,
          max(a12.day_date)  day_date,
          sum(a11.sale_extended_amount)  WJXBFS1
from      dbo.vf_sale_txn_1503        a11
          join      dbo.vm_day        a12
            on      (a11.dk_vm_sale_date = a12.pk_vm_day)
where     a11.dk_vm_sale_date in (10889, 10925, 10958)
group by a11.dk_vm_sale_date
```

As you can see the SQL generated goes to only the three Manually Partitioned Fact Tables necessary to retrieve the data for the query. It applies the constraints for the days to each of the three sub queries because it does not know which days co-relate to which fact tables.

However, the point to be made clear is that MicroStrategy, not the SQL Server Optimiser, has selected the fact tables from which to select data.

The use of the Union All will make the queries that much easier to understand and tune.

This SQL being generated is as opposed to creating a view that is union alled and then applying constraints and letting the optimiser decide which tables to read. When we were testing this sort of partitioning where the tables were unioned before being presented to MicroStrategy we found that the SQL Server Optimiser had some significant trouble identifying the partitions to read from the queries being provided.

As we tested the various ways in which Manual Partitioning could be presented to MicroStrategy we decided that it would be best if we allowed MicroStrategy to generate the union statements and to decide what partitions to use to perform the queries.

7. SUMMARY

This section summarises what has been presented in this white paper.

It is widely agreed, and supported by MicroStrategy, that when building a data warehouse some form of Partitioning Strategy should be used to:

- Improve Query Performance
- Improve ETL Performance
- Reduce overall Machine utilisation
- Reduce maintenance operations for tables
- Reduce the time taken for DBAs to support the database.

It is widely accepted that to implement a data warehouse without some form of Partitioning Strategy when there are sizable tables involved (100M+ rows) is a significant error.

Many MicroStrategy customers use SQL Server for their data warehouse database.

In SQL Server database supported partitioning was introduced in SQL Server 2005 and it was enhanced in 2008 and 2008R2. It was only ever made available in Enterprise Edition.

When MicroSoft released SQL Server 2012 Enterprise Edition licensing rules were changed so that the CAL license was no longer available for SQL Server 2012 Enterprise Edition. Only core licenses were made available.

This has left some MicroStrategy + SQL Server users stuck on 2008R2 because of the cost of the upgrade to 2012 Enterprise Edition to retain the database implementation of the Partitioning Features.

Some MicroStrategy + SQL Server customers made the mistake of not using some form of Partitioning.

This white paper presents an example of implementing Manual Partitioning using SQL Server 2014 Standard Edition in conjunction with the MicroStrategy Partition Mapping Feature.

By using this example a MicroStrategy + SQL Server customer can choose to implement Manual Partitioning in SQL Server Standard Edition, or similar, for 2012, 2014, 2016. They can do this and not have to pay the higher license fees for the Enterprise Edition.

Such an implementation will deliver the benefits of Database Level Partitioning for the very reduced cost of some DBA work to set up the tables that are required and to configure MicroStrategy to make use of this Manual Partitioning.

We present this White Paper to the MicroStrategy + SQL Server customer community to give all such companies the opportunity to learn from us as to how to perform such an implementation to improve the overall performance of you MicroStrategy Implementation.

In this white paper we have provided the information to create a Manually Partitioned Fact Table for the BIDA Sales Transaction Fact Table for a 3 year period, 2015, 2016, 2017.

We have provided the following information.

- Setting up File Groups for the Tables and Indexes
- Setting up the Files for the Table and Indexes
- Setting up the Tables for the Manually Partitioned Tables.
- Setting up the Primary Keys for the tables.
- Setting up the Constraints for the Partitioning Column for the tables.
- Setting up the MicroStrategy Partitioning Schema Table
- Setting up the MicroStrategy Partitioning Schema View
- Importing the Partitioning Schema View in to the Warehouse Manager
- Creating the Attributes needed to make the partitioning work
- Creating the Partitioning Attribute
- Creating the Attributes to make a simple report work
- Creating a single fact and metric to make a simple report work
- Creating a simple report to show Extended Sales Amount by day
- The SQL that was generated by MicroStrategy showing only necessary tables were queried

You can obtain the source code for all of the above from us to set up this demonstration for yourself if you wish to do so.

Naturally, if you are a MicroStrategy Consulting Company and you do not have strong SQL Server Skills we would be pleased to work with you and your customers to implement and support such Manual Partitioning so that you can deliver improved query performance for your customers.

We hope that by making this information freely available more users of MicroStrategy will be able to implement Partitioning on SQL Server so that they will improve the performance of their data warehouses for their customers.

If you would like to contact us please just email Iulian Lixandru on Iulian@empowerbi.com.

We would be very happy to talk with you about Manual Partitioning and what we have been able to achieve in our clients by using these techniques.