|  |  |
| --- | --- |
| C:\Documents and Settings\stevepon\Desktop\ofc-PrfrmPt-2_rgb.png | TER_4C_TAG.tif |

# Accessing large scale ROLAP environments with PerformancePoint Server 2007

Rupal Shah

Teradata Corporation

Steve Pontello

Microsoft Corporation

Published: December 2007

Applies To: PerformancePoint Monitoring Server 3.0

Summary: Monitoring Server is part of Microsoft Office PerformancePoint Server 2007. This white paper is for PerformancePoint Server administrators and IT professionals who want guidance about how to develop and deploy PerformancePoint Monitoring Server applications utilizing ROLAP data sources built with Analysis Services and Teradata.

### Copyright

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

 2007 Microsoft Corporation. All rights reserved.

Microsoft, Windows, and Windows Serverare either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

All other trademarks are property of their respective owners.

### Table of Contents

**Introduction** **4**

**Building a Demonstration Dashboard** **5**

Defining the Data Source 6

Defining the Prior Period Sales Growth KPI 7

Creating the Prior Period Growth Scorecard 10

Creating Analytic Reports 13

Creating the Sales Dashboard 15

Previewing the Sales Dashboard 18

Summary of the Dashboard Designer Experience 19

**Ad-hoc Analysis and Throughput Testing** **21**

Testing Methodology 21

Results 21

Appendix A – PerformancePoint Monitoring Server Architecture 22

Appendix B – Server Configuration 23

Appendix C – Project Real Teradata Source Tables 24

Appendix D – Teradata Database Tuning Considerations 27

Appendix E – Throughput Testing Results 30

#### **Introduction**

In any business, aligning employee activities to company objectives can be a challenging task. An integrated approach to performance management can provide greater insight into business performance with less cost and effort than traditional solutions. With Microsoft® Office PerformancePointTM Server 2007, your business can effectively monitor performance against company goals; analyze key data in a way that provides rapid insight into trends, patterns, opportunities and challenges in your business environment; and plan for the business’ future direction and results, while helping you better anticipate internal and external factors that can impact the business. By enabling a continual process of monitoring, analysis and planning, your business becomes more effective, agile and able to better take advantage of quickly changing market conditions.

This document describes how to create and render reports using a ROLAP data source built with Analysis Services and Teradata. In addition, KPIs can be built using an ODBC data source that accesses data residing in the Teradata data source utilizing a SQL statement.

The following areas are covered by an existing whitepaper titled: “Improve your OLAP Environment with Microsoft and Teradata”. To access the whitepaper, go to:

<http://www.microsoft.com/downloads/details.aspx?FamilyID=8f38fafb-a7c3-4cc1-a059-7a7a732058a2&DisplayLang=en>

* How do I build a ROLAP cube using Analysis Services and Teradata?
* What is the role of the Analysis Services developer and the Teradata database administrator?
* How can I improve ROLAP performance using the Teradata Aggregate Join Index (AJI) feature?

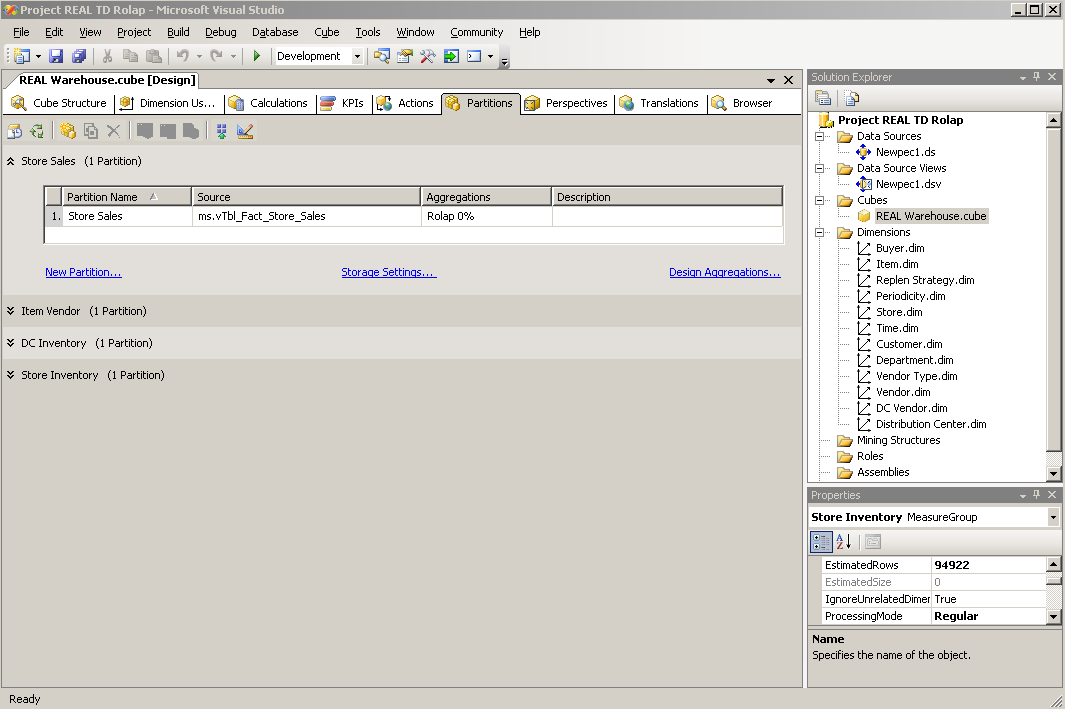
The following “how do I” solutions are addressed in this whitepaper:

* Building a simple dashboard using the PerformancePoint Dashboard Designer.
* Performing ad-hoc analysis with chart and grid reports.

#### **Building a Demonstration Dashboard**

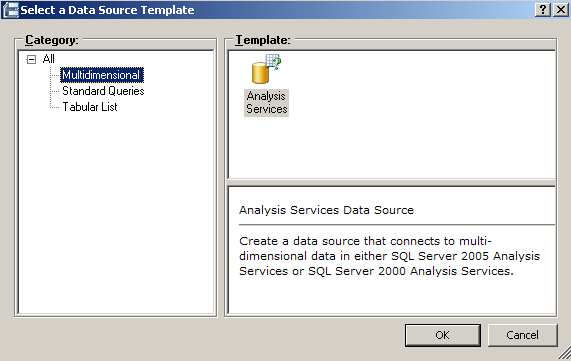
The ROLAP source used for the demonstration dashboard was built utilizing data from Project Real. For additional information about Project Real, go to:

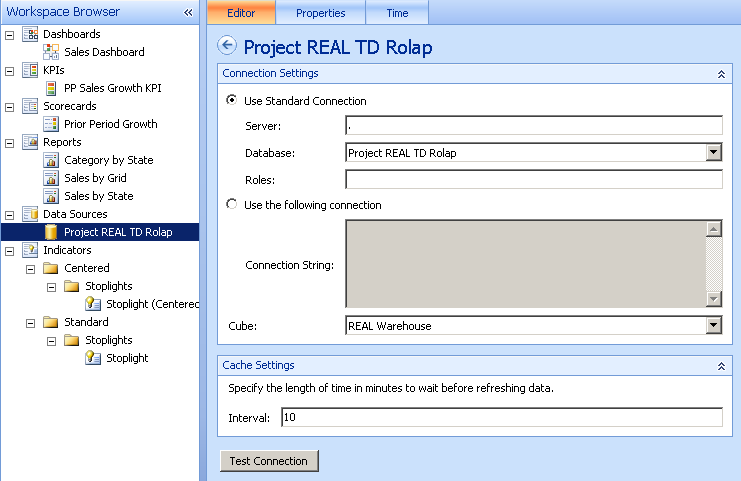
<http://www.microsoft.com/sql/solutions/bi/projectreal.mspx>

The primary fact table (view) used as the source for the *Real Warehouse* cube was *vTbl\_Fact\_Store\_Sales* which contained 727,011,130 rows of data (see [Appendix C](#_Appendix_C_–)). The cube partition used the ROLAP storage mode with 0% aggregation. Query performance against the source Teradata table benefitted greatly from a Teradata Aggregate Join Index (AJI) built utilizing the fact table and supporting dimension tables (see [Appendix D](#_Appendix_D_–)).

## Defining the Data Source

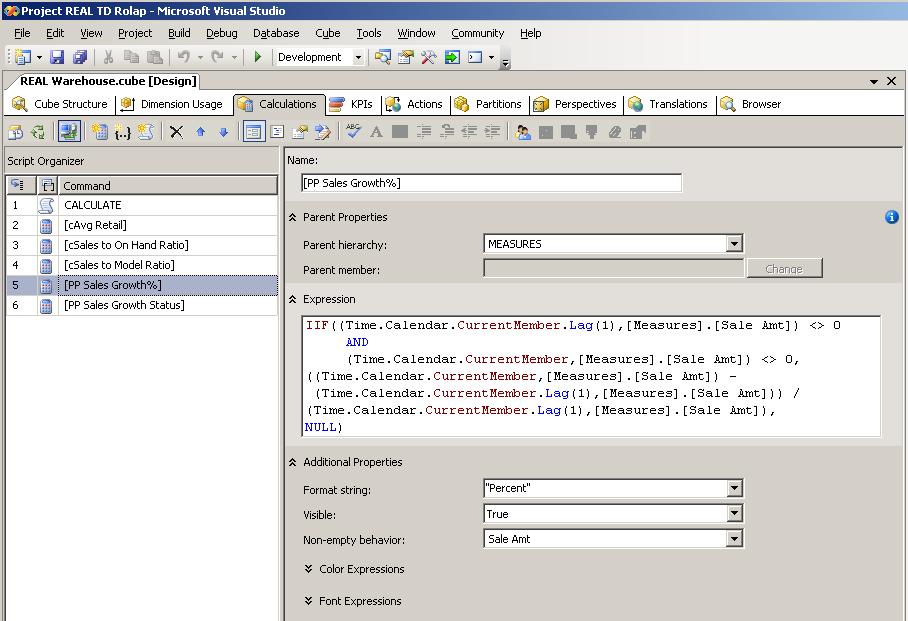
The data source utilized for analytic reports, scorecard and dashboard was created using the standard *Multidimensional – Analysis Services* data source template.





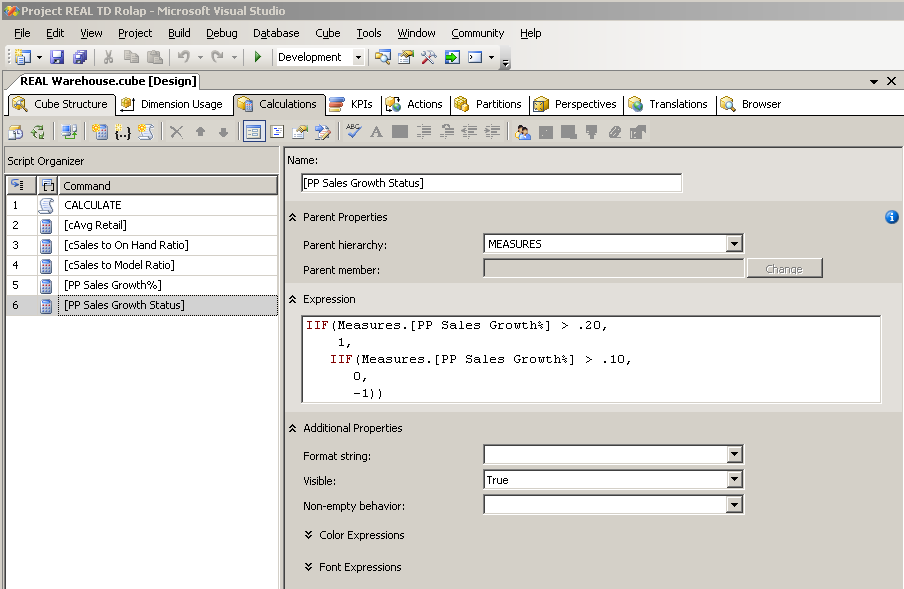
## Defining the Prior Period Sales Growth KPI

The *Prior Period Sales Growth KPI* was built using Business Intelligence Development Studio (BIDS) and leveraged cube script calculations for the *Value Expression* and the *Status Expression*.

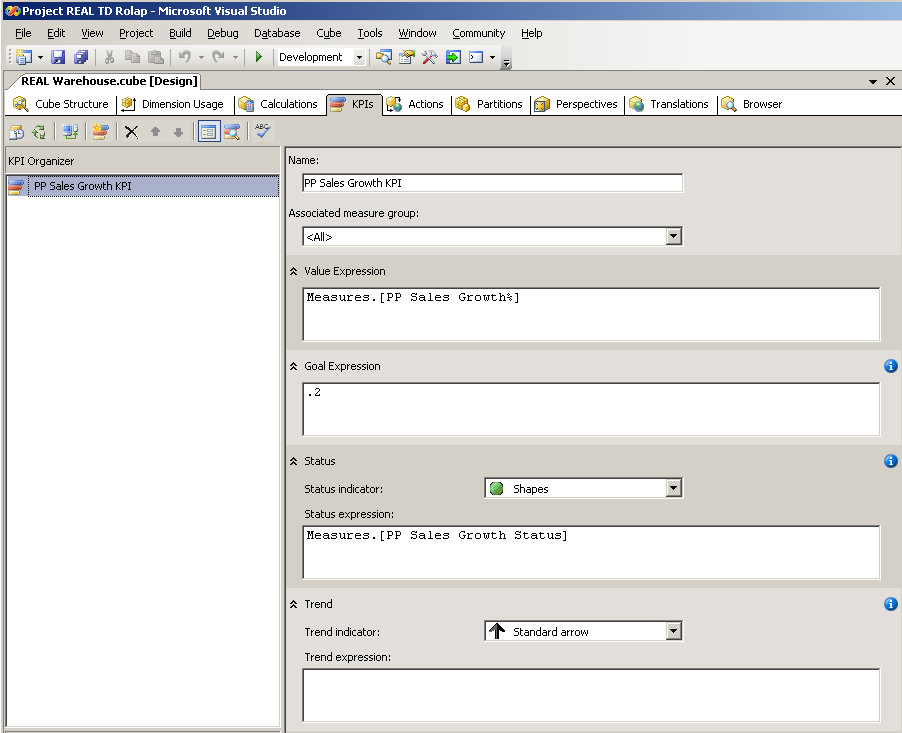
PP Sales Growth% Calculated Member Definition:

The *PP Sales Growth%* is calculated by subtracting the previous period sales amount from the current period sales amount and then dividing the result by the previous period sales amount. The MDX leverages the *CurrentMember* function to return the correct result based on the time period selected by the end user. If the end user selects a year, then the calculation will compare the total for the year against the total for the previous year. If the end user selects a month, then the calculation will compare the total for the month against the total for the previous month.

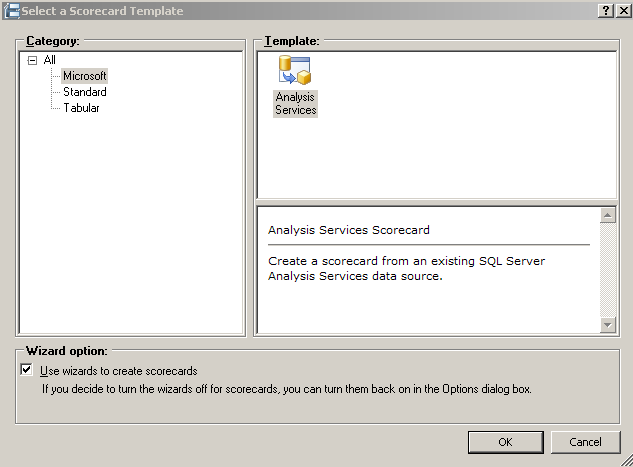
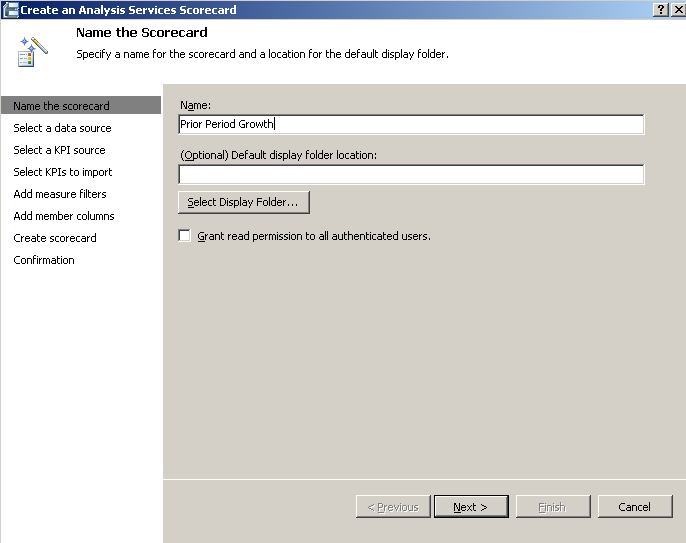
PP Sales Growth Status Calculated Member Definition:

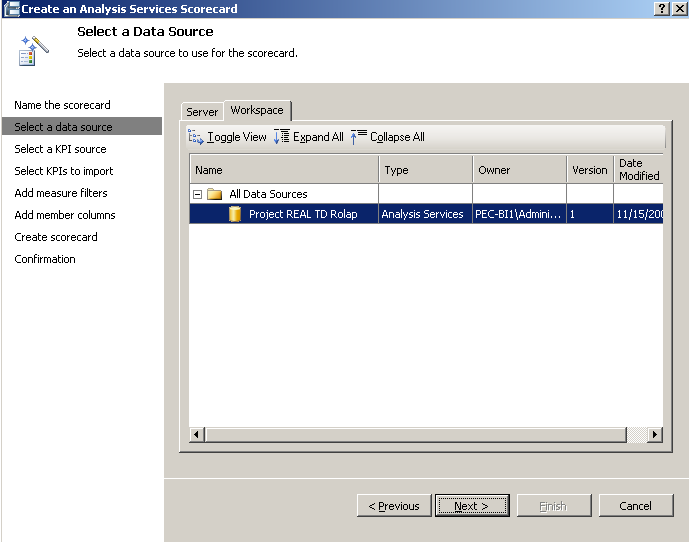
The definition will set the status indicator for the *PP Sales Growth KPI* to be *Green* if the sales growth % is greater than 20%, *Yellow* if the sales growth % is between 10% and 20% and *Red* otherwise.

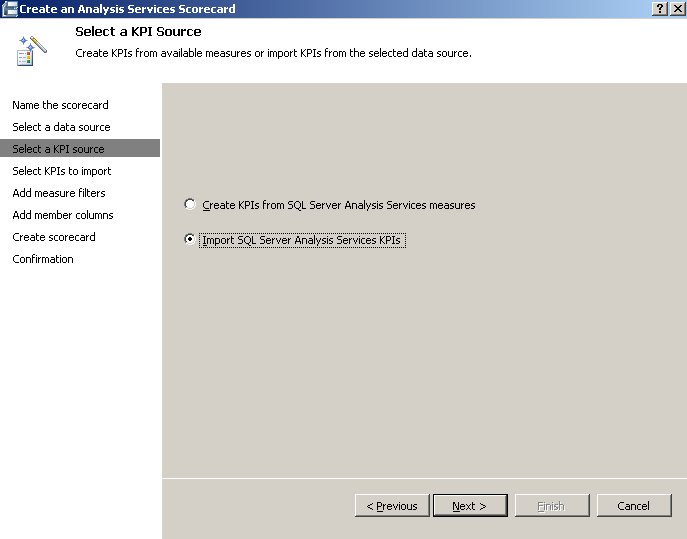
PP Sales Growth KPI Definition:

The *Goal Expression* has been set to the fixed value of 20%. There is no *Trend Expression* for this KPI.

## Creating the Prior Period Growth Scorecard

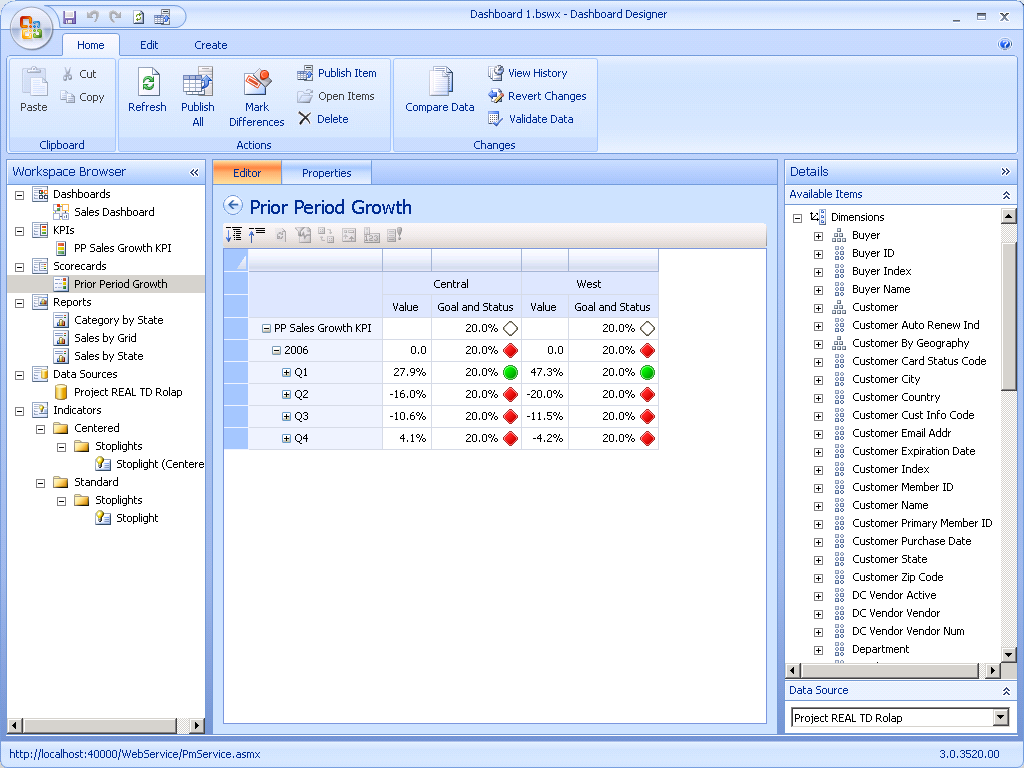
The *Prior Period Growth* scorecard was built using the ROLAP data source and importing the *PP Sales Growth KPI* defined in the previous step.





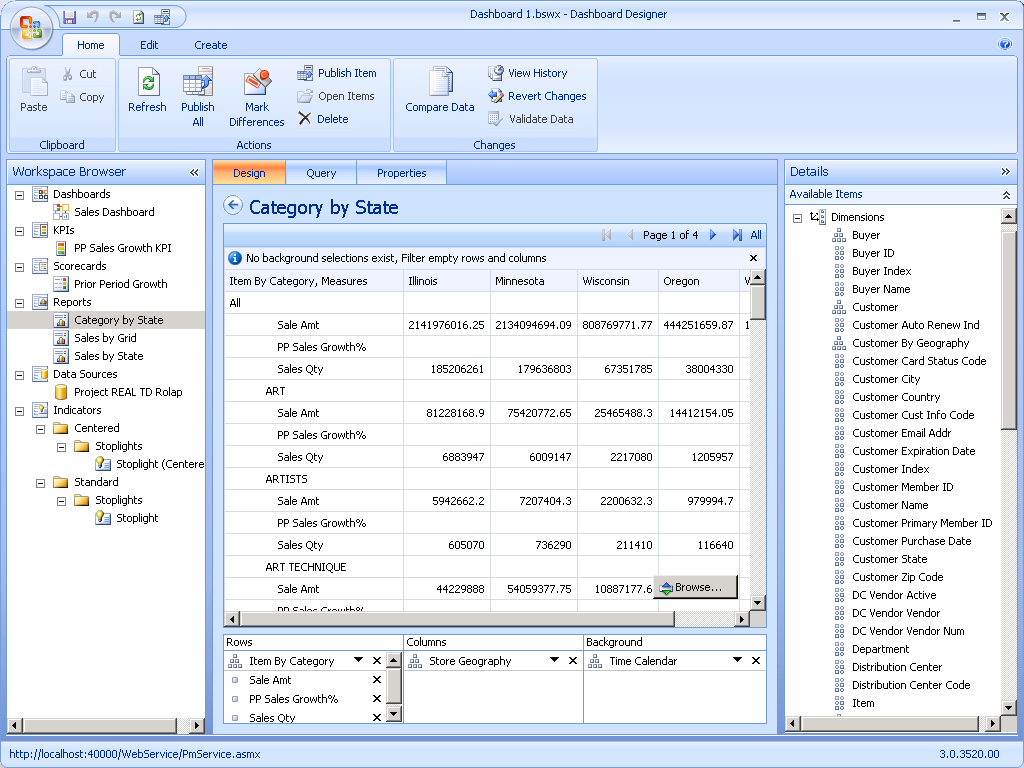
In this step the existing KPI definition contained in the *Real Warehouse* cube is imported to create the scorecard.

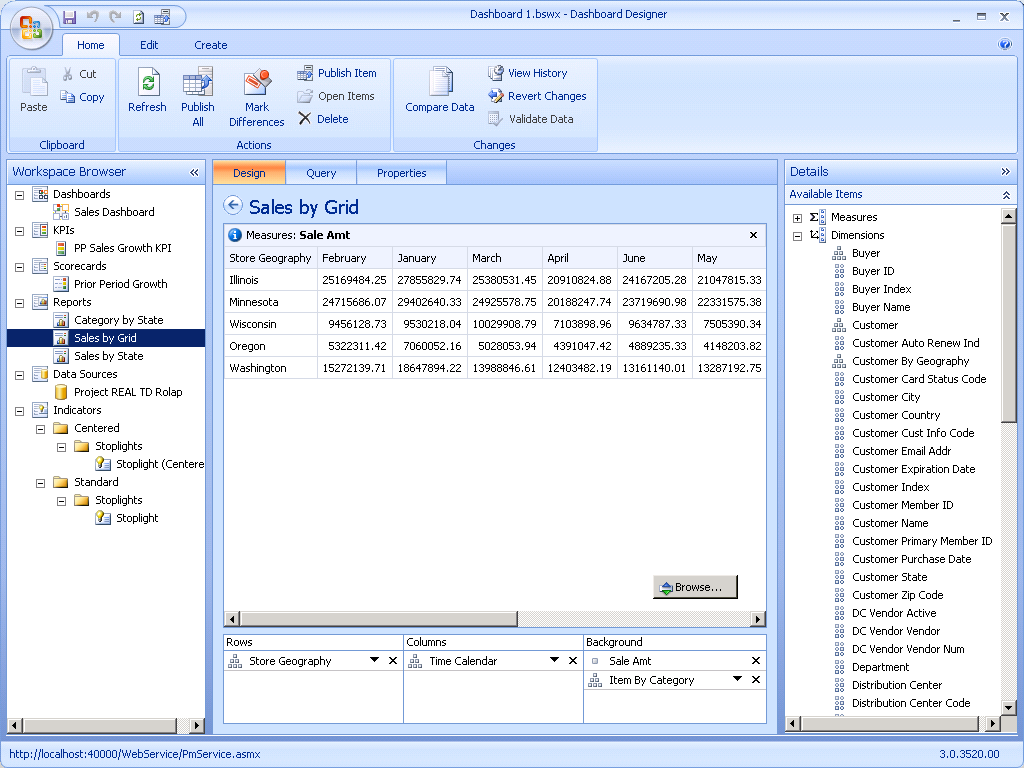
The scorecard is completed by dragging the *Central* and *West* regions from the *Store Geography* hierarchy onto the column headers and *2006* along with its quarters on to rows from the *Time Calendar* hierarchy.

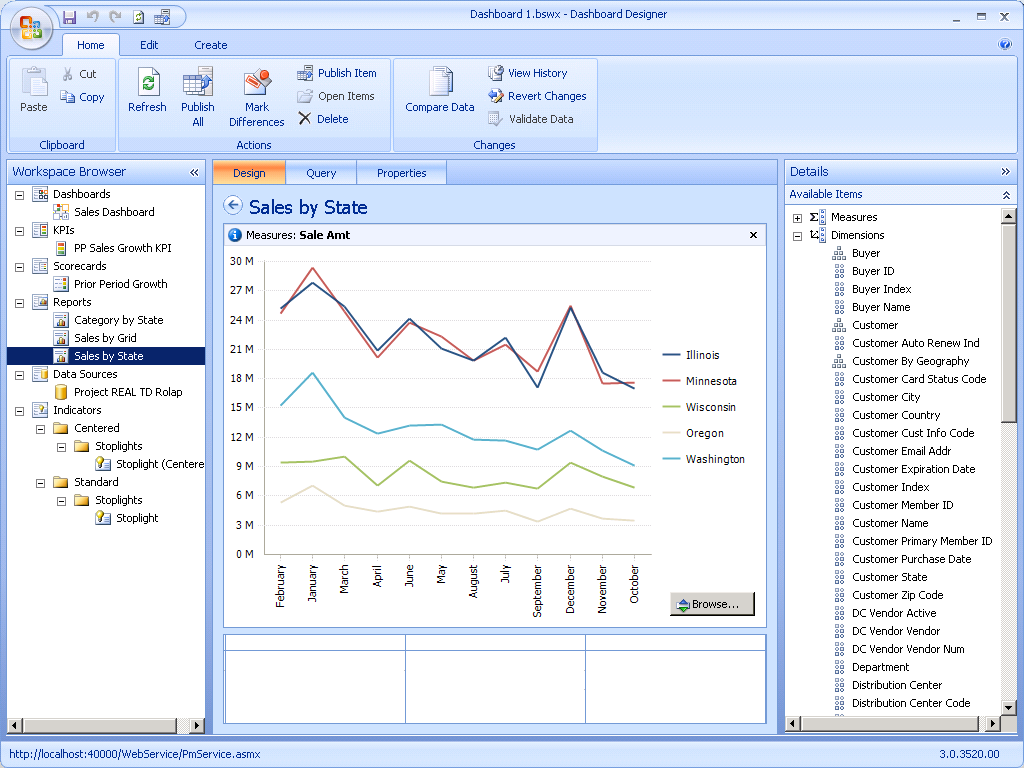


## Creating Analytic Reports

Three analytic reports were defined to be included in the demonstration dashboard. All three reports leveraged the *Project REAL TD ROLAP* data source. The reports were created by simply dragging and dropping the appropriate hierarchies on to Rows, Columns and Background.

Category by State analytic grid report:

Sales by Store Geography analytic grid:



Sales by Store Geography analytic chart:

## Creating the Sales Dashboard

The *Sales Dashboard* was created using the scorecard and three analytic view reports created in the previous steps. In addition, three filters were created to filter the analytic view reports.

Creating the Filters

The three dashboard filters were all created using MDX statements to define the member sets used to populate the filters.

Product Categories Filter

|  |  |
| --- | --- |
| MDX | [Item].[By Category].[Category].Members |
| Display Method | List |

Time Multi Filter

|  |  |
| --- | --- |
| MDX | Descendants(Time.Calendar.DefaultMember,  Time.Calendar.[Calendar Month],  SELF\_AND\_BEFORE) |
| Display Method | Multi-Select Tree |

Time 2006 Filter

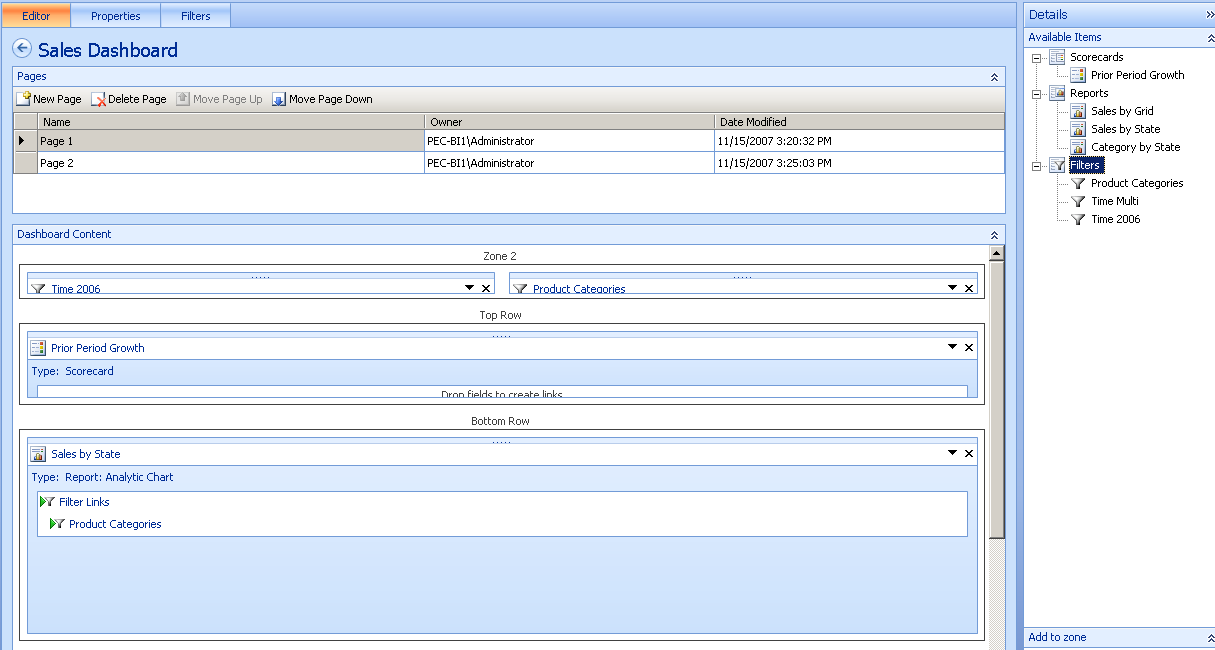
|  |  |
| --- | --- |
| MDX | Descendants([Time].[Calendar].[Calendar Year].&[2006],  Time.Calendar.[Calendar Month],  SELF\_AND\_BEFORE) |
| Display Method | Tree |

Creating the Dashboard Pages

The dashboard is comprised of two pages. Each page demonstrates different filter and report types. The dashboard pages were created by dragging and dropping filter and reports into page zones. The filters where then linked to the appropriate targets on each analytic view report.

Page 1

|  |  |
| --- | --- |
| Filters | Time 2006, Product Categories |
| Scorecards | Prior Period Growth |
| Reports | Sales by State (filtered by Product Categories),  Sales by State Grid (filtered by Product Categories and Time 2006) |

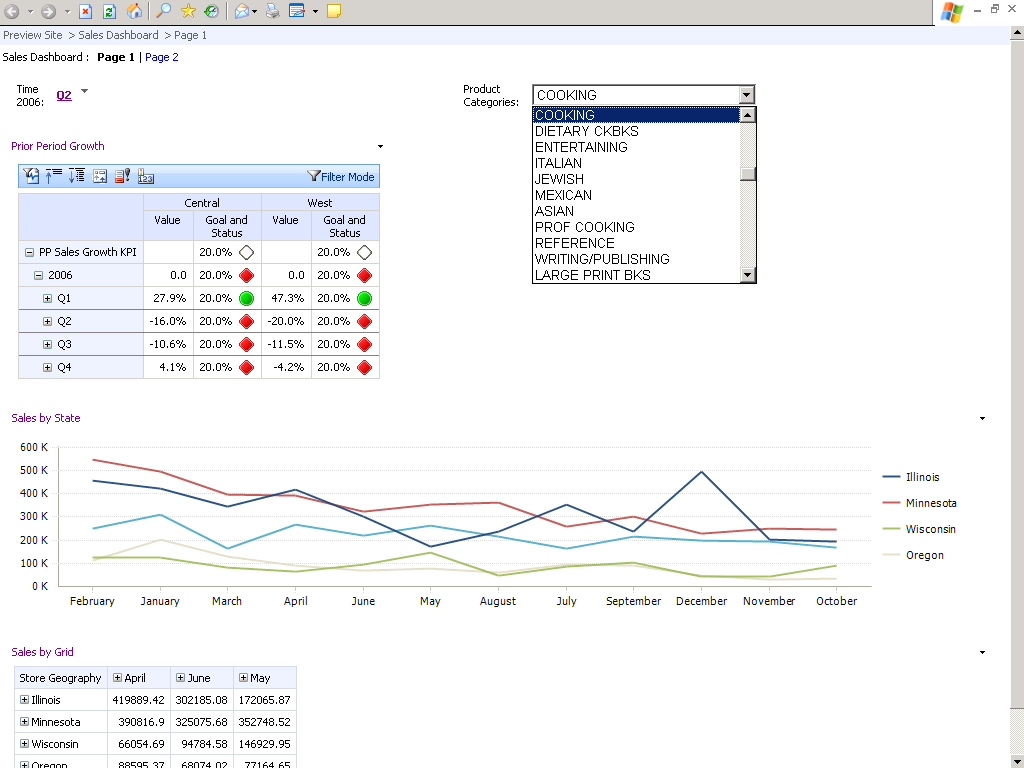
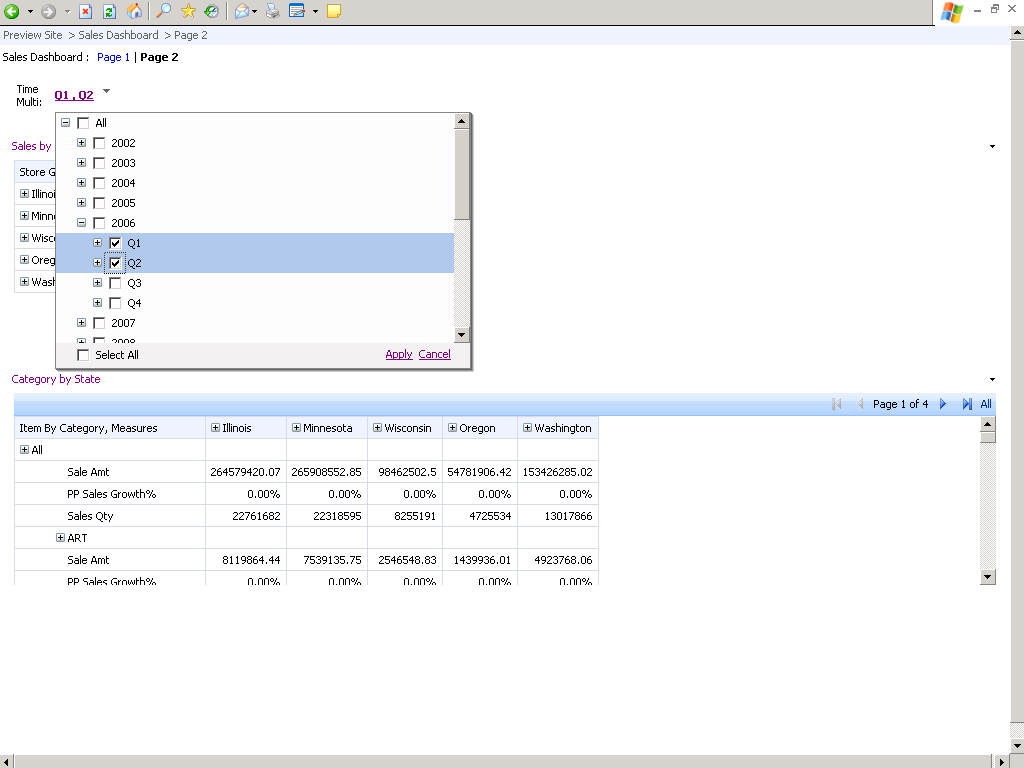


Page 2

|  |  |
| --- | --- |
| Filters | Time Multi |
| Scorecards | None |
| Reports | Sales by State Grid (filtered by Time Multi)  Category by State (filtered by Time Multi) |



## Previewing the Sales Dashboard

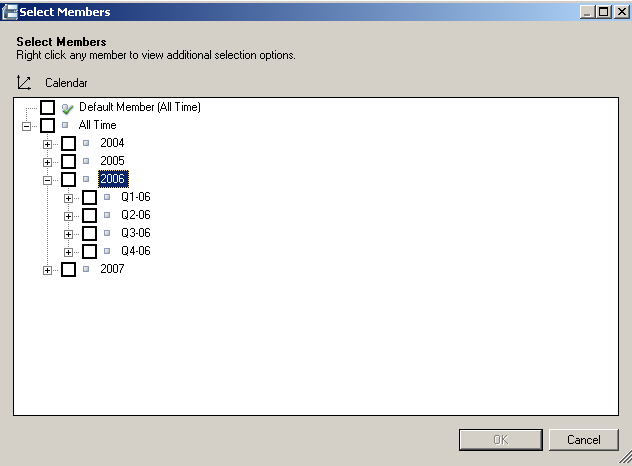
Upon completion the *Sales Dashboard* was viewed using the *Preview* feature of the Dashboard Designer.

## Summary of the Dashboard Designer Experience

During the build phase of the *Sales Dashboard* the Dashboard Designer executes many small MDX queries to populate the design interface and provide previews of the objects being developed. For example, if the developer chooses to select the quarters for the year *2006* then a query is sent to the Analysis Services data source to retrieve the children for that member. Analysis Services in turn will issue a SQL query against Teradata to verify data exists for the children requested. In most cases these queries return instantly, but some queries can require more time to return if members are derived from a level outside of the scope of the AJI or fact table level are requested. In which case, standard table indexes (i.e. secondary or partitioned primary indexes) can also be implemented to improve performance of queries generated by the Dashboard Designer. Therefore, it is important when defining an indexing strategy for large fact tables to take into account the performance of the design phase queries, as well as, queries executed during analysis of the data by a business analyst.

**Example**

Expanding the year 2006 in the Dashboard Designer results in the following MDX and SQL queries being executed:



|  |  |
| --- | --- |
| MDX | SELECT { AddCalculatedMembers({[Time].[Calendar].[Calendar Year].&[2006].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| SQL | SELECT SUM ( "ms\_vTbl\_Fact\_Store\_Sales"."Sales\_Qty" ) AS "ms\_vTbl\_Fact\_Store\_Sales0\_0", SUM ( "ms\_vTbl\_Fact\_Store\_Sales"."Sale\_Amt" ) AS "ms\_vTbl\_Fact\_Store\_Sales0\_1","ms\_vTbl\_Dim\_Date\_4"."Calendar\_Qtr\_ID" AS "ms\_vTbl\_Dim\_Date1\_0","ms\_vTbl\_Dim\_Date\_4"."Calendar\_Year\_ID" AS "ms\_vTbl\_Dim\_Date1\_1" FROM "ms"."vTbl\_Fact\_Store\_Sales" AS "ms\_vTbl\_Fact\_Store\_Sales","ms"."vTbl\_Dim\_Date" AS "ms\_vTbl\_Dim\_Date\_4" WHERE ( ( "ms\_vTbl\_Fact\_Store\_Sales"."SK\_Date\_ID" = "ms\_vTbl\_Dim\_Date\_4"."SK\_Date\_ID" ) AND ( "ms\_vTbl\_Dim\_Date\_4"."Calendar\_Year\_ID" = 2006 ) ) GROUP BY "ms\_vTbl\_Dim\_Date\_4"."Calendar\_Qtr\_ID","ms\_vTbl\_Dim\_Date\_4"."Calendar\_Year\_ID" |

During the development of the *Sales Dashboard* there was only one occurrence of a long running metadata query issued by the Dashboard Designer. This occurred when the members of the Customer Geography -> County Code members were requested. This was expected, since the Customer Geography dimension was not included in the AJI definition which would be tied to the *vTbl\_Fact\_Store\_Sales* as the source for the cube measures (see [Appendix – D](#_Appendix_D_–)).

#### **Ad-hoc Analysis and Throughput Testing**

The current version of PerformancePoint Monitoring Server allows end users to analyze data either within the Dashboard or by launching a new window from the dashboard containing an individual scorecard or analytic view report. Scorecards can be expanded/collapsed, Filtered, or modified to show different rollups (average weighted and worst child). Analytic views provide analysts with the ability to filter empty rows/columns, expand/collapse, change view type (bar chart, stacked bar chart, line chart, or grid), drill down/up, sort (using compact or tabular layout options), pivot and cross-drill to other levels or attribute hierarchies. In addition, the ability to *drill to detail* and execute *cell level actions* is available using the analytic view report type. The *drill to detail* will retrieve row level detail for the referenced cell from the fact table and associated dimension tables. The *cell level actions* are defined within the Analysis Services cube and give the analytic application developer the ability to call other tools, such as Reporting Services, to provide additional insight to the number displayed in a cell.

## Testing Methodology

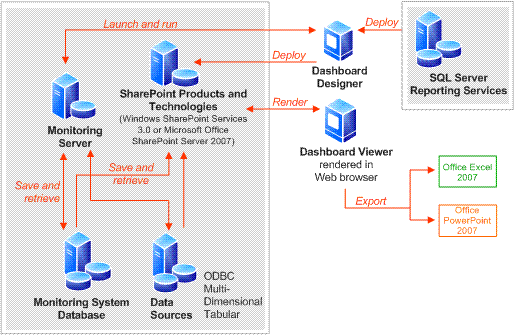
The MDX queries from two separate “analysts” sessions were captured using SQL Profiler to be used for throughput testing. The first recorded analyst session generated 115 distinct MDX queries. The second recorded analyst session generated 197 distinct MDX queries. Throughput testing was conducted by running concurrent batch files from command line windows. The batch files used ASCMD to cycle through all of the queries captured from the analysis sessions. There were 5 batches running the 115 queries from the first analyst session and 5 batches running the 197 queries from the second analyst session. During the testing all 10 batch windows were started within 10 seconds. SQL Profiler was used to capture queries and query response time during the testing.

## Results

The longest running query from the throughput testing ran in 375 milliseconds (see samples of longest running queries in [Appendix – E](#_Appendix_E_–)). Some additional testing was done to evaluate *drill to detail* functionality and analysis of members outside the scope of the AJI defined on the Teradata fact table view. The drill to detail ran in under 3 minutes returning the first 1000 rows of detail fact table data.

Additional testing was performed to ensure that if an end user cancelled a query that appropriate response would occur on both Analysis Services and Teradata. The results of the testing confirmed any query stopped from an analytic view resulted in the halting of the MDX query on Analysis Services and the corresponding SQL query on the Teradata server.

# Appendix A – PerformancePoint Monitoring Server Architecture

The PerformancePoint Monitoring Server is comprised of multiple components that communicate with each other during the development, deployment and rendering of dashboards, scorecards and reports. Some components are registered with other Microsoft software products, such as SharePoint Server and Business Intelligence Design Studio. For a complete reference describing each component, go to: <http://technet.microsoft.com/en-us/library/bb821220.aspx>

**Figure 1   PerformancePoint Monitoring Server component architecture**

# Appendix B – Server Configuration

Analysis Services and PerformancePoint Server 2007 were both installed on the same server for this demonstration project.

|  |  |
| --- | --- |
| System | DELL quad-core system |
| Processor | Single 1.86 GHz Intel Xeon x64 64-bit |
| RAM | 4 GB |
| Disk | 1 TB |
| OS | Windows Server 2003 x64 |
| Software | Microsoft SQL Server 2005 x64 (Enterprise Edition)  Microsoft Office PerformancePoint Server 2007 x64 |

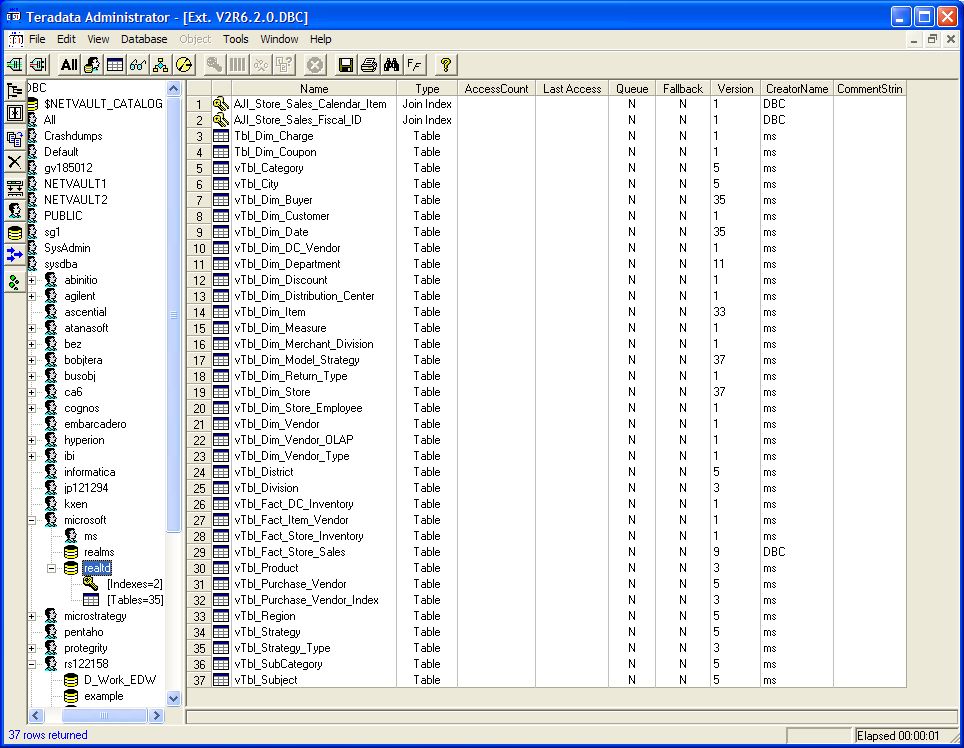
Note:

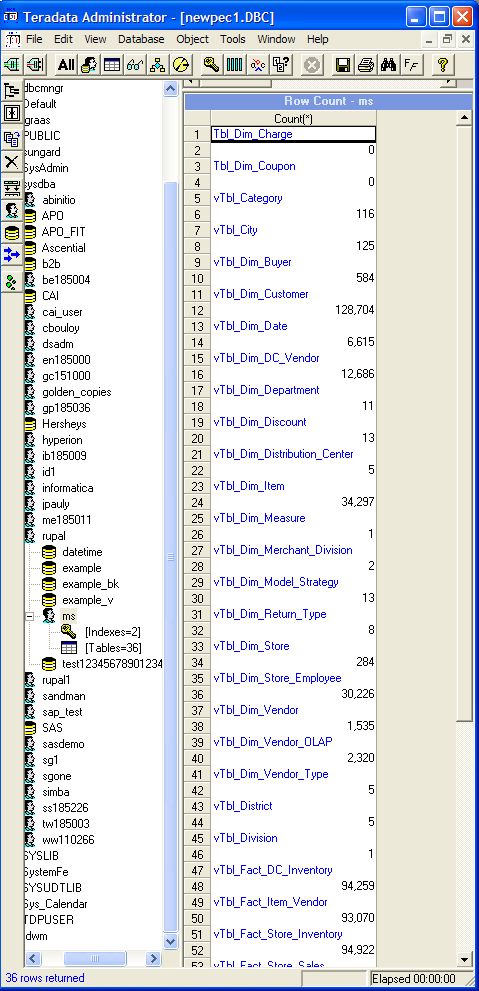
The ROLAP architecture described in this document greatly reduces the amount of disk resources required by Analysis Services. In a Teradata ROLAP scenario, a best practice is to use MOLAP storage for dimensions and ROLAP storage with no aggregations for measure group partitions (The Teradata Aggregate Join Indexes are used to ensure excellent query response time).

The Teradata Database server used for this demonstration.

|  |  |
| --- | --- |
| System | 4 Node Teradata 5500 Sytem |
| Each Node | 12 AMPs and 2 PEs |
| 4 CPUs @ 2.66 Ghz |
| 8 GB RAM |
| Linux/SUSE9 (64-bit) |
| Disk | RAID1 with 96 x 73 GB disk drives |
| Software | Teradata Database version V2R6.2.1.3 |

# Appendix C – Project Real Teradata Source Tables





# cid:105345617@26112007-07FB

# Appendix D – Teradata Database Tuning Considerations

The following is a summary of the activities performed to prep for this engagement. The exact order and level of parallel effort may not be directly expressed in this summary.

Database Preparation:

* Migration of the sample data from the Project REAL kit into a Teradata system provided by Teradata. This was accomplished at the Microsoft Technology Center in Chicago through remote access to a Teradata system that was located at the Rancho Bernardo Center in San Diego. Used Teradata Administration tools to migrate the schema and SQL Server Integration Services to load the data.
* Identified a set of Report Services reports for this activity and with these reports, identified the tables and columns that were being queried.
* Migrated the Project REAL Star Schema into a Snow-Flake and tuned it for the Teradata System. Star schemas would perform well, but would depend on volume of data, cardinality and size of Teradata Database system. As well as, customer objectives and Aggregate Join Index (AJI)[[1]](#footnote-2) design for analytics. For our activities, we performed the following database tuning and considerations mentioned below:
* Created additional tables to support snowflake dimensions
* Ensured all dimension table primary keys are defined as unique utilizing the UNIQUE constraint, or the primary key is defined as a UNIQUE PRIMARY INDEX.
* Ensured all UNIQUE PRIMARY INDEX (UPI) columns are defined with NOT NULL
* Ensured (where possible) all primary and foreign keys are on the ID and not Name or Description columns. This will result in a smaller AJI, which means faster data access.
* Ensured single level dimensions have supporting reference/lookup/dimension table for optimal performance.
* Populated the snow-flake dimension tables with data (i.e. insert/select)
* Ensured Fact Table design is ‘wide’[[2]](#footnote-3)(i.e., columns for each dimension and measure) and modified Primary Index (PI) as a composite key of all non-measure columns
* Collect statistics on all primary key/foreign key relationship columns.
* Implement Referential Integrity (RI) on the primary key/foreign key columns. RI can be defined with the check (a.k.a., hard RI) or no check option (a.k.a., soft RI).

*A Snow-Flake schema with the tuning mentioned above will result in smaller AJI which can produce faster query response versus a Star schema which may result in less database tuning activities, but will require all columns of interest to be included in the AJI definition. Customers should do their own due diligence to determine the best schema design and database tuning based on their objectives.*

* Redesigned the Project REAL Analysis Services Solution on the Snow-Flake schema while maintaining a consistent matching database structure to facilitate the query requirements of the Reporting Services reports.
* The following dimension map identifies the Reporting Services reports and the AJI strategy/performance requirements.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TIME | STORE | ITEM | | | DEPARTMENT | BUYER | REPLEN STRATEGY | MEASURE |
| Year\_Desc | Division | Product | Purchase Vendor Index | | Department | Buyer Alpha | Strategy Type | Sales\_Qty |
| Qtr\_Desc | Region | Subject | Purchase Vendor | |  | Buyer Name | Strategy | Sale\_Amt |
| Month\_Desc | District | Category | |  |  |  | Title Type | . |
| Week\_Desc | City | Sub Category | |  |  |  | . |
| Date\_Value | Store\_  Desc | Item\_Desc | |  |  |  | . |

* Built 1 ‘broad’ [[3]](#footnote-4)AJI to support/cover the above (in yellow):

CREATE JOIN INDEX ms.AJI\_Store\_Sales ,NO FALLBACK ,CHECKSUM = DEFAULT AS

SELECT COUNT(\*)(FLOAT, NAMED CountStar ),

ms.f.Buyer\_Alpha ,

ms.a.SK\_Dept\_ID ,

ms.e.Strategy\_Ind ,

ms.b.Purch\_Vendor\_Num ,

ms.b.Dept ,

ms.b.Subcategory\_Code ,

ms.c.City ,

ms.d.Calendar\_Year\_Desc ,

ms.d.Calendar\_Qtr\_Desc ,

ms.d.Calendar\_Month\_Desc ,

ms.d.Calendar\_Week\_Desc ,

SUM(ms.a.Sales\_Qty )(FLOAT, NAMED Sales\_Qty ),

SUM(ms.a.Sale\_Amt )(FLOAT, NAMED Sale\_Amt )

FROM

ms.vtbl\_fact\_store\_sales a ,

ms.vtbl\_dim\_item b ,

ms.vtbl\_dim\_store c ,

ms.vtbl\_dim\_date d ,

ms.vtbl\_dim\_model\_strategy e ,

ms.vtbl\_dim\_buyer f

WHERE

((((ms.a.SK\_Item\_ID = ms.b.SK\_Item\_ID ) AND

(ms.a.SK\_Store\_ID = ms.c.SK\_Store\_ID )) AND

(ms.a.SK\_Date\_ID = ms.d.SK\_Date\_ID )) AND

(ms.a.SK\_Model\_Strategy\_ID = ms.e.SK\_Model\_Strategy\_ID )) AND

(ms.a.SK\_Buyer\_ID = ms.f.Sk\_Buyer\_ID )

GROUP BY

ms.f.Buyer\_Alpha ,

ms.a.SK\_Dept\_ID ,

ms.e.Strategy\_Ind ,

ms.b.Purch\_Vendor\_Num ,

ms.b.Dept ,

ms.b.Subcategory\_Code ,

ms.c.City ,

ms.d.Calendar\_Year\_Desc ,

ms.d.Calendar\_Qtr\_Desc ,

ms.d.Calendar\_Month\_Desc ,

ms.d.Calendar\_Week\_Desc

PRIMARY INDEX

(Buyer\_Alpha ,SK\_Dept\_ID ,Strategy\_Ind ,Purch\_Vendor\_Num ,

Dept ,Subcategory\_Code ,City ,Calendar\_Year\_Desc ,Calendar\_Qtr\_Desc ,

Calendar\_Month\_Desc ,Calendar\_Week\_Desc );

# Appendix E – Throughput Testing Results

|  |  |
| --- | --- |
| **Milliseconds** | **MDX Statement** |
| 375 | WITH MEMBER [Measures].[CUSTOM\_8e1f18cc-a99e-4d21-9943-8f6b4091c69e\_0b5bbdb2-674d-4003-a6cb-0b74e466eaf0] as '(KPIGoal("PP Sales Growth KPI"))' MEMBER [Measures].[8e1f18cc-a99e-4d21-9943-8f6b4091c69e\_0b5bbdb2-674d-4003-a6cb-0b74e466eaf0] as '([Measures].[CUSTOM\_8e1f18cc-a99e-4d21-9943-8f6b4091c69e\_0b5bbdb2-674d-4003-a6cb-0b74e466eaf0])' MEMBER [Measures].[CUSTOM\_8e1f18cc-a99e-4d21-9943-8f6b4091c69e\_33cbf01d-1683-4e77-8297-43c4dcfc4bb9] as '(KPIStatus("PP Sales Growth KPI"))' MEMBER [Measures].[8e1f18cc-a99e-4d21-9943-8f6b4091c69e\_33cbf01d-1683-4e77-8297-43c4dcfc4bb9] as '([Measures].[CUSTOM\_8e1f18cc-a99e-4d21-9943-8f6b4091c69e\_33cbf01d-1683-4e77-8297-43c4dcfc4bb9])' MEMBER [Measures].[CUSTOM\_8e1f18cc-a99e-4d21-9943-8f6b4091c69e\_d4f783e1-53e2-4a84-9848-be0e76b2df9a] as '(KPIValue("PP Sales Growth KPI"))' MEMBER [Measures].[8e1f18cc-a99e-4d21-9943-8f6b4091c69e\_d4f783e1-53e2-4a84-9848-be0e76b2df9a] as '([Measures].[CUSTOM\_8e1f18cc-a99e-4d21-9943-8f6b4091c69e\_d4f783e1-53e2-4a84-9848-be0e76b2df9a])'SELECT { [Measures].[8e1f18cc-a99e-4d21-9943-8f6b4091c69e\_0b5bbdb2-674d-4003-a6cb-0b74e466eaf0],[Measures].[8e1f18cc-a99e-4d21-9943-8f6b4091c69e\_33cbf01d-1683-4e77-8297-43c4dcfc4bb9],[Measures].[8e1f18cc-a99e-4d21-9943-8f6b4091c69e\_d4f783e1-53e2-4a84-9848-be0e76b2df9a]} ON AXIS(0),{([Store].[Geography].[Region].&[2.],[Time].[Calendar].[Calendar Month].&[2]&[2006]),([Store].[Geography].[Region].&[3.],[Time].[Calendar].[Calendar Month].&[2]&[2006]),([Store].[Geography].[Region].&[2.],[Time].[Calendar].[Calendar Month].&[1]&[2006]),([Store].[Geography].[Region].&[3.],[Time].[Calendar].[Calendar Month].&[1]&[2006]),([Store].[Geography].[Region].&[2.],[Time].[Calendar].[Calendar Month].&[3]&[2006]),([Store].[Geography].[Region].&[3.],[Time].[Calendar].[Calendar Month].&[3]&[2006]),([Store].[Geography].[Region].&[2.],[Time].[Calendar].[Calendar Month].&[4]&[2006]),([Store].[Geography].[Region].&[3.],[Time].[Calendar].[Calendar Month].&[4]&[2006]),([Store].[Geography].[Region].&[2.],[Time].[Calendar].[Calendar Month].&[6]&[2006]),([Store].[Geography].[Region].&[3.],[Time].[Calendar].[Calendar Month].&[6]&[2006]),([Store].[Geography].[Region].&[2.],[Time].[Calendar].[Calendar Month].&[5]&[2006]),([Store].[Geography].[Region].&[3.],[Time].[Calendar].[Calendar Month].&[5]&[2006]),([Store].[Geography].[Region].&[2.],[Time].[Calendar].[Calendar Month].&[8]&[2006]),([Store].[Geography].[Region].&[3.],[Time].[Calendar].[Calendar Month].&[8]&[2006]),([Store].[Geography].[Region].&[2.],[Time].[Calendar].[Calendar Month].&[7]&[2006]),([Store].[Geography].[Region].&[3.],[Time].[Calendar].[Calendar Month].&[7]&[2006]),([Store].[Geography].[Region].&[2.],[Time].[Calendar].[Calendar Month].&[9]&[2006]),([Store].[Geography].[Region].&[3.],[Time].[Calendar].[Calendar Month].&[9]&[2006]),([Store].[Geography].[Region].&[2.],[Time].[Calendar].[Calendar Month].&[12]&[2006]),([Store].[Geography].[Region].&[3.],[Time].[Calendar].[Calendar Month].&[12]&[2006]),([Store].[Geography].[Region].&[2.],[Time].[Calendar].[Calendar Month].&[11]&[2006]),([Store].[Geography].[Region].&[3.],[Time].[Calendar].[Calendar Month].&[11]&[2006]),([Store].[Geography].[Region].&[2.],[Time].[Calendar].[Calendar Month].&[10]&[2006]),([Store].[Geography].[Region].&[3.],[Time].[Calendar].[Calendar Month].&[10]&[2006])} ON AXIS(1)FROM [REAL Warehouse] |
| 203 | SELECT { AddCalculatedMembers({[Store].[Geography].[District].&[1.7E1].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 203 | SELECT { AddCalculatedMembers({[Item].[By Category].[Sub Category].&[010AB].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 188 | SELECT { AddCalculatedMembers({[Store].[Geography].[District].&[5.6E1].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 188 | SELECT { [Item].[By Category].[Category].Members } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 172 | SELECT { AddCalculatedMembers({[Replen Strategy].[Replen Strategy].[Strategy Type].&[1].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 172 | SELECT { AddCalculatedMembers({[Store].[Geography].[District].&[5.8E1].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 172 | SELECT { AddCalculatedMembers({[Item].[By Category].[Category].&[010A].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 172 | SELECT { AddCalculatedMembers({[Store].[Geography].[District].&[2.8E1].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 156 | SELECT { AddCalculatedMembers({[Store].[Geography].[District].&[4.3E1].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 141 | SELECT { AddCalculatedMembers({[Time].[Calendar].[Calendar Month].&[11]&[2006].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 125 | SELECT { DISTINCT({[Time].[Calendar].[Calendar Month].&[4]&[2006].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 109 | SELECT { DISTINCT({[Time].[Calendar].[Calendar Month].&[7]&[2006].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 109 | SELECT { AddCalculatedMembers({[Time].[Calendar].[Calendar Month].&[1]&[2006].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 109 | SELECT { AddCalculatedMembers({[Time].[Calendar].[Calendar Qtr].&[4]&[2006].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 109 | SELECT { AddCalculatedMembers({[Time].[Calendar].[Calendar Month].&[2]&[2006].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 109 | SELECT { Descendants([Time].[Calendar].[Calendar Year].&[2006],Time.Calendar.[Calendar Month],SELF\_AND\_BEFORE) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 109 | SELECT { Descendants(Time.Calendar.DefaultMember,Time.Calendar.[Calendar Month],SELF\_AND\_BEFORE) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 109 | SELECT { DISTINCT({[Time].[Calendar].[Calendar Month].&[12]&[2006].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |
| 109 | SELECT { AddCalculatedMembers({[Time].[Calendar].[Calendar Month].&[3]&[2007].Children}) } DIMENSION PROPERTIES MEMBER\_TYPE on 0 FROM [REAL Warehouse] |

1. An AJI is an aggregated result set saved as an index in the database. It is transparent to an end-user and BI Administrators and will be used automatically by the Teradata optimizer when a query plan contains frequently made like columns and aggregates. Refer to the whitepaper mentioned earlier in this paper for more details. [↑](#footnote-ref-2)
2. A ‘wide’ Fact table design has the measure and dimension columns in it representing the lowest level of detail with the dimension columns being used in the Composite Primary Key of the fact table. In addition, the non-measured columns are used for RI as foreign key to supporting dimension table unique primary key. Refer to the whitepaper mentioned earlier in this paper for more details. [↑](#footnote-ref-3)
3. The ‘broad’ AJI constitutes as an index which covers all dimensions and levels of interest based on the aforementioned physical database stipulations to effectively address multi level dimensional hierarchy/model. Refer to the whitepaper mentioned earlier in this paper for more details. [↑](#footnote-ref-4)