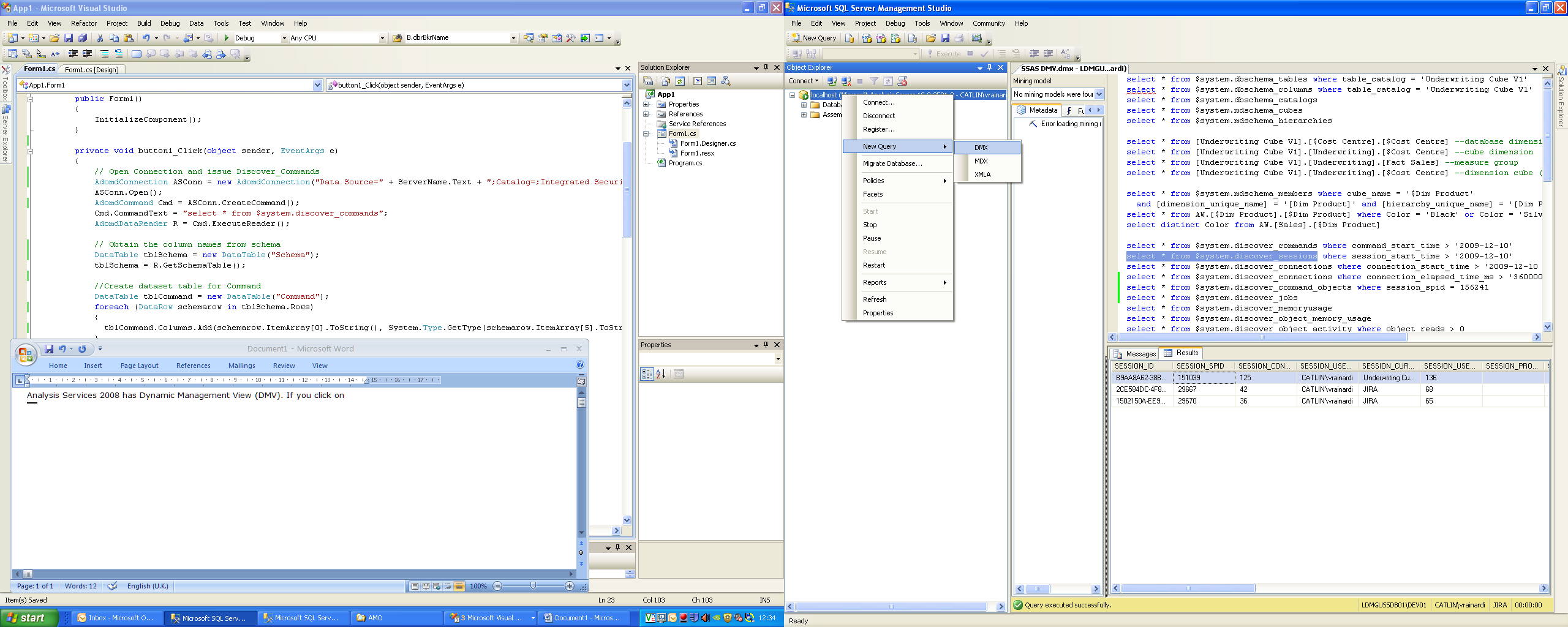
SSAS DMV: Join Using DataSet

The purpose of this program is to provide an example, how DMV can be loaded into DataSet, and displayed on DataGridView, to provide administrative information about an SSAS server. Basically any DMV can be use this way, opening the door to use it as ‘cube structure browser’ as well as an admin tool.

The Visual Studio project is created using VS2010 Beta 2, as Win app. The description below is taken from [my blog post](http://dwbi1.wordpress.com/2009/12/29/ssas-dmv-join-using-dataset/).

In SSAS 2008 we have Dynamic Management View (DMV), which is basically XMLA discover command, wrapped as data mining select statement.

On SSMS, if we connect to an SSAS instance, right click on the server name, New Query, DMX, type “select \* from $system.discover\_connections”, and click Execute, we will see a list of users currently connecting to that server.



The output is like this: (I only pick 1 row and transpose it so we can see the column names clearly)

|  |  |
| --- | --- |
| CONNECTION\_ID | 42 |
| CONNECTION\_USER\_NAME | domain\user1 |
| CONNECTION\_IMPERSONATED\_USER\_NAME |  |
| CONNECTION\_HOST\_NAME | 10.1.27.146:3315 |
| CONNECTION\_HOST\_APPLICATION | Microsoft SQL Server Management Studio - Query |
| CONNECTION\_START\_TIME | 29/12/2009 11:34 |
| CONNECTION\_ELAPSED\_TIME\_MS | 3089784 |
| CONNECTION\_LAST\_COMMAND\_START\_TIME | 29/12/2009 11:45:03 |
| CONNECTION\_LAST\_COMMAND\_END\_TIME | 29/12/2009 11:45:04 |
| CONNECTION\_LAST\_COMMAND\_ELAPSED\_TIME\_MS | 62 |
| CONNECTION\_IDLE\_TIME\_MS | 1972821 |
| CONNECTION\_BYTES\_SENT | 3610612 |
| CONNECTION\_DATA\_BYTES\_SENT | 15880694 |
| CONNECTION\_BYTES\_RECEIVED | 1502120 |
| CONNECTION\_DATA\_BYTES\_RECEIVED | 1502120 |

And if we type “select \* from $system.discover\_sessions” we will get:

|  |  |
| --- | --- |
| SESSION\_ID | D1A8209F-27C4-4013-AB23-154704213CB3 |
| SESSION\_SPID | 357640 |
| SESSION\_CONNECTION\_ID | 661 |
| SESSION\_USER\_NAME | domain\user1 |
| SESSION\_CURRENT\_DATABASE | Cube1 |
| SESSION\_USED\_MEMORY | 71 |
| SESSION\_PROPERTIES |  |
| SESSION\_START\_TIME | 29/12/2009 11:57 |
| SESSION\_ELAPSED\_TIME\_MS | 3193141 |
| SESSION\_LAST\_COMMAND\_START\_TIME | 29/12/2009 12:09 |
| SESSION\_LAST\_COMMAND\_END\_TIME | 29/12/2009 12:09 |
| SESSION\_LAST\_COMMAND\_ELAPSED\_TIME\_MS | 0 |
| SESSION\_IDLE\_TIME\_MS | 2476374 |
| SESSION\_CPU\_TIME\_MS | 87687 |
| SESSION\_LAST\_COMMAND | {MDX Statement} |
| SESSION\_LAST\_COMMAND\_CPU\_TIME\_MS | 62 |
| SESSION\_STATUS | 0 |
| SESSION\_READS | 103571 |
| SESSION\_WRITES | 0 |
| SESSION\_READ\_KB | 6593264 |
| SESSION\_WRITE\_KB | 0 |
| SESSION\_COMMAND\_COUNT | 1854 |

And when we type “select \* from $system.discover\_commands” we get:

|  |  |
| --- | --- |
| SESSION\_SPID | 356127 |
| SESSION\_COMMAND\_COUNT | 31 |
| COMMAND\_START\_TIME | 29/12/2009 11:36 |
| COMMAND\_ELAPSED\_TIME\_MS | 0 |
| COMMAND\_CPU\_TIME\_MS | 281 |
| COMMAND\_READS | 102 |
| COMMAND\_READ\_KB | 6154 |
| COMMAND\_WRITES | 0 |
| COMMAND\_WRITE\_KB | 0 |
| COMMAND\_TEXT | {MDX Statement} |
| COMMAND\_END\_TIME | 29/12/2009 11:36 |

For other DMVs please see my June presentation at Microsoft TVP here.

We can link the discover\_sessions and discover\_connections on CONNECTION\_ID. And we can link the discover sessions and discover\_commands on SESSION\_SPID. Like this:



If we want to join the three DMVs (command, session, connection), we have 3 options:

1. Linked Server
2. SSIS
3. .NET DataSet

Midas Matelis [wrote](http://www.ssas-info.com/VidasMatelisBlog/144_using-ssrs-to-report-ssas-2008-database-structure-using-dmvs#more-144) about the Linked Server. Basically it’s using OpenQuery.  
Chris Webb [wrote](http://sqlserverpedia.com/blog/sql-server-bloggers/killing-sessions-automatically-with-ssis/) about the SSIS one.  
This article is about using DataSet to join the DMVs. It’s using ADOMD. So we need to add reference to Microsoft.AnalysisServices.AdomdClient.

What is a DataSet? We can think of DataSet as in-memory ERD diagram. It’s a collection of tables which have relational integrity between them.

So, we are going to issue Discover\_Connections, Sessions and Commands then join them in the DataSet. The steps are:

1. Add reference to ADOMDClient
2. Open a connection and execute Discover\_Commands
3. Create a DataTable for Discover\_Commands and populate it
4. Execute Discover\_Sessions
5. Create a DataTable for Discover\_Sessions and populate it
6. Execute Discover\_Connections
7. Create a DataTable for Discover\_Connections and populate it
8. Add all three tables to the DataSet and join them
9. Display the joined table to the screen

1. Add reference to ADOMDClient

Create a new C# project. You can use Windows form, WPF or web project. Or Silverlight project if you use VS2010. Add reference to Microsoft.AnalysisServices.AdomdClient

2. Open a connection and execute Discover\_Commands

AdomdConnection ASConn = new AdomdConnection("Data Source=ServerName;Catalog=;Integrated Security=SSPI;");

ASConn.Open();

AdomdCommand Cmd = ASConn.CreateCommand();

Cmd.CommandText = "select \* from $system.discover\_commands";

AdomdDataReader R = Cmd.ExecuteReader();

Leave the Catalog empty. We don’t need to specify it if we only want to execute discover command, session and connection. We do need to specify the database name for some DMV.

3. Create a DataTable for Discover\_Commands and populate it

First we create a DataTable called tblCommand:

DataTable tblCommand = new DataTable("Command");

Then we get the column names from the schema:

DataTable tblSchema = new DataTable("Schema");

tblSchema = R.GetSchemaTable();

foreach (DataRow schemarow in tblSchema.Rows)

{ tblCommand.Columns.Add(schemarow.ItemArray[0].ToString(), System.Type.GetType(schemarow.ItemArray[5].ToString()));}

Then we populate the DataTable from the ADOMD Data Reader:

while (R.Read())

{object[] ColArray = new object[R.FieldCount];

for (int i = 0; i < R.FieldCount; i++)

{if (R[i] != null) ColArray[i] = R[i];}

tblCommand.LoadDataRow(ColArray, true);

}

R.Close();

4. Execute Discover\_Sessions

We reuse the data reader.

Cmd.CommandText = "select \* from $system.discover\_sessions";

R = Cmd.ExecuteReader();

5. Create a dataset table for Discover\_Sessions and populate it

First we create a DataTable called tblSession:

DataTable tblSession = new DataTable("Session");

Then we get the column names from the schema:

tblSchema = R.GetSchemaTable();

foreach (DataRow schemarow in tblSchema.Rows)

{tblSession.Columns.Add(schemarow.ItemArray[0].ToString(), System.Type.GetType(schemarow.ItemArray[5].ToString()));}

Then we populate the DataTable from the ADOMD Data Reader:

while (R.Read())

{object[] ColArray = new object[R.FieldCount];

for (int i = 0; i < R.FieldCount; i++)

{if (R[i] != null) ColArray[i] = R[i]; }

tblSession.LoadDataRow(ColArray, true);

}

R.Close();

6. Execute Discover\_Connections (reuse reader)

Cmd.CommandText = "select \* from $system.discover\_connections";

R = Cmd.ExecuteReader();

7. Create a DataTable for Discover\_Connections and populate it

First we create a DataTable called tblConnection:

DataTable tblConnection = new DataTable("Connection");

Get the column names from schema (tblSchema is defined above)

Column name is on the 1st column, data type is on the 6th column.

tblSchema = R.GetSchemaTable();

foreach (DataRow schemarow in tblSchema.Rows)

{ //column name is on the 1st column, data type is on the 6th column

tblConnection.Columns.Add(schemarow.ItemArray[0].ToString(),

System.Type.GetType(schemarow.ItemArray[5].ToString()));

}

Then we populate the Connection DataTable

while (R.Read())

{

object[] ColArray = new object[R.FieldCount];

for (int i = 0; i < R.FieldCount; i++)

{

if (R[i] != null) ColArray[i] = R[i];

}

tblConnection.LoadDataRow(ColArray, true);

}

Then we close the data reader and the connection because we have finished reading data from SSAS. After this we only play in memory.

R.Close();

ASConn.Close(true);

ASConn.Dispose();

8. Add all three tables to the dataset and join them

Create a DataSet and add the data tables:

DataSet DS = new DataSet();

DS.Tables.Add(tblCommand);

DS.Tables.Add(tblSession);

DS.Tables.Add(tblConnection);

Create a relationship between Session and Command on Session SPID

DataColumn ParentColumn = DS.Tables["Session"].Columns["SESSION\_SPID"];

DataColumn ChildColumn = DS.Tables["Command"].Columns["SESSION\_SPID"];

DataRelation Rel = new DataRelation("Session\_Command", ParentColumn, ChildColumn);

DS.Tables["Command"].ParentRelations.Add(Rel);

Create a relationship between Connection and Session on Connection ID

ParentColumn = DS.Tables["Connection"].Columns["CONNECTION\_ID"];

ChildColumn = DS.Tables["Session"].Columns["SESSION\_CONNECTION\_ID"];

DataRelation Rel2 = new DataRelation("Connection\_Session", ParentColumn, ChildColumn);

DS.Tables["Session"].ParentRelations.Add(Rel2);

Join the three tables: (David M provides a good [reference](http://weblogs.sqlteam.com/davidm/archive/2004/01/20/748.aspx), using array)

First we create a DataTable called tblJoin to store the final result:  
DataTable tblJoin = new DataTable(“Join”);

Then we create the columns of tblJoin, by getting the column names and data types from tblConnection, tblSession and tblCommand. To avoid duplicate column name, add “2″ on the second join column.

DataTable tblJoin = new DataTable("Join");

for (int i = 0; i < tblConnection.Columns.Count; i++)

{

tblJoin.Columns.Add(tblConnection.Columns[i].ColumnName, tblConnection.Columns[i].DataType);

}

for (int i = 0; i < tblSession.Columns.Count; i++)

{

if (!tblJoin.Columns.Contains(tblSession.Columns[i].ColumnName))

tblJoin.Columns.Add(tblSession.Columns[i].ColumnName, tblSession.Columns[i].DataType);

else

tblJoin.Columns.Add(tblSession.Columns[i].ColumnName + "2", tblSession.Columns[i].DataType);

}

for (int i = 0; i < tblCommand.Columns.Count; i++)

{

if (!tblJoin.Columns.Contains(tblCommand.Columns[i].ColumnName))

tblJoin.Columns.Add(tblCommand.Columns[i].ColumnName, tblCommand.Columns[i].DataType);

else

tblJoin.Columns.Add(tblCommand.Columns[i].ColumnName + "2", tblCommand.Columns[i].DataType);

}

tblJoin.BeginLoadData();

foreach (DataRow ConnectionRow in tblConnection.Rows)

{

DataRow[] SessionRows = ConnectionRow.GetChildRows(Rel2);

if (SessionRows != null && SessionRows.Length > 0)

{

object[] ConnectionArray = ConnectionRow.ItemArray;

foreach (DataRow SessionRow in SessionRows)

{

DataRow[] CommandRows = SessionRow.GetChildRows(Rel);

if (CommandRows != null && CommandRows.Length > 0)

{

object[] SessionArray = SessionRow.ItemArray;

foreach (DataRow CommandRow in CommandRows)

{

object[] CommandArray = CommandRow.ItemArray;

object[] JoinArray = new object[ConnectionArray.Length

+ SessionArray.Length + CommandArray.Length];

Array.Copy(ConnectionArray, 0, JoinArray, 0, ConnectionArray.Length);

Array.Copy(SessionArray, 0, JoinArray, ConnectionArray.Length, SessionArray.Length);

Array.Copy(CommandArray, 0, JoinArray, ConnectionArray.Length +

SessionArray.Length, CommandArray.Length);

tblJoin.LoadDataRow(JoinArray, true);

}

}

}

}

}

7. Display the joined table to the screen

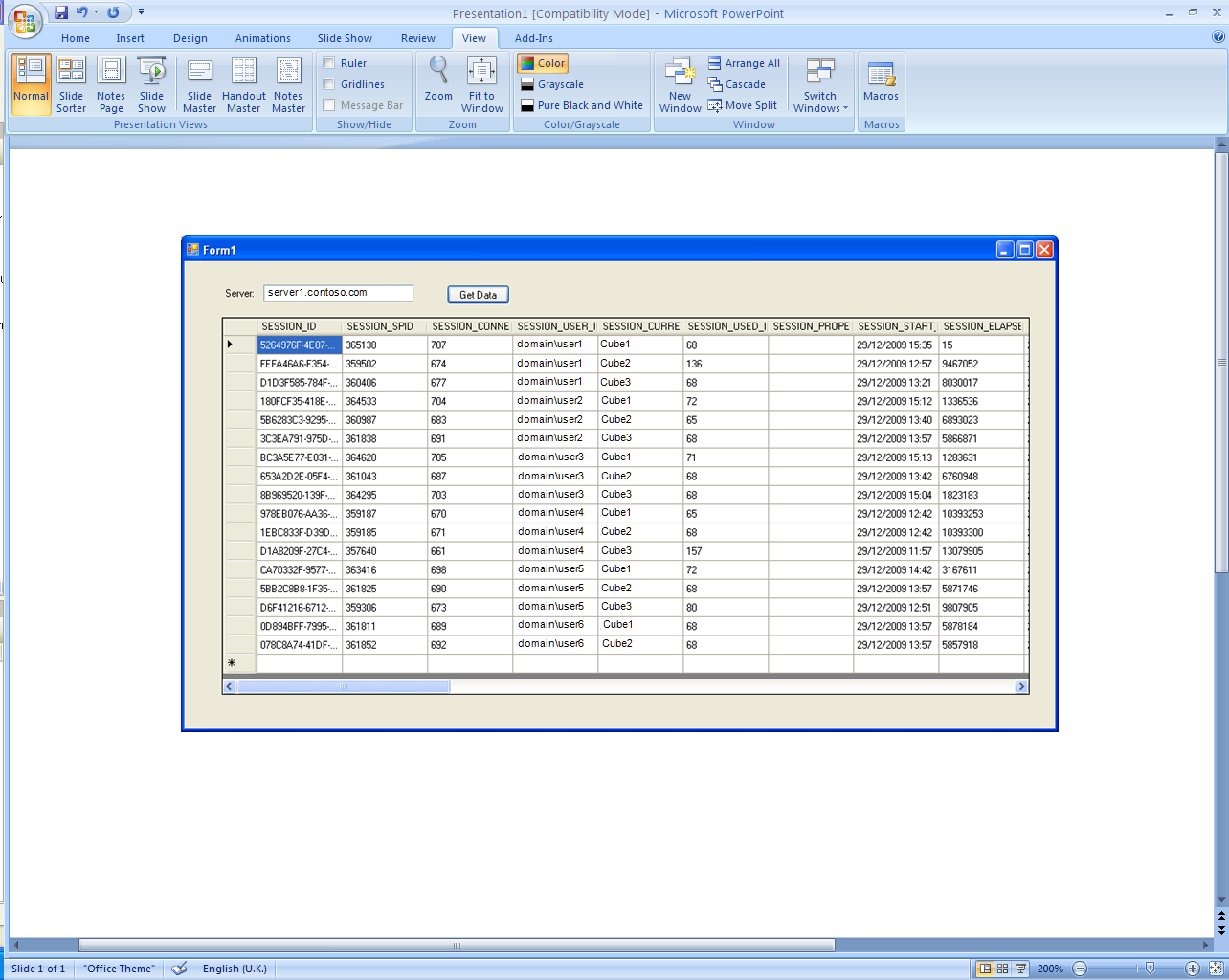
Here I used DataGridView object. If you want the AutoSizeColumnsMode can be set to cover the widest content.

dataGridView1.DataSource = tblJoin;

We can also display only the column that we like (not all columns).

We can put the server name in a text box so we can change SSAS server easily.

The result contains 48 columns: (the tblConnection columns are not shown)



Potential for further development:

1. Automatic refresh every N seconds, store in history table (not the duplicate rows)
2. Create a data mart from the history table on point 1 (fact & dim tables), then load into cube. Similar to my approach for OlapQueryLog cube.
3. Cascade drill down: you click + on a connection and it displays the session, you click + on a connection and it displays the commands.
4. Execute other DMVs, such as displaying object resources (memory etc) used by each command or connection.
5. Automatically give warning by email if the amount of resources used by a connection exceeds certain pre-set limits.