# SQL_R2_Logo.jpg

SQL Server Technical Article

# Enabling and Securing Data Entry with Analysis Services Writeback

**Writers:** Pablo Trejo Montemayor (Microsoft), Sergei Gundorov (Microsoft)

**Technical Reviewer:** Irina Gorbach

**Published:** December 2010

**Applies to:** SQL Server 2008 and SQL Server 2008 R2

**Summary:** Analysis Services is a tool for aggregating information and providing business users with the ability to analyze and support decision making in their business. By using the built-in writeback feature in Analysis Services, business users can also modify their data points to perform what-if analysis or supplement any existing data.

This article captures several best practices that can help you use writeback more effectively. The techniques described in this article derive from the author’s professional experience in the design and development of complex financial analysis applications used by various business groups in a large multinational company. In this article, the authors share their techniques for managing dynamic security, managing large data volume in the writeback table, and setting up connections in a way that improves writeback performance.

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

© 2010 Microsoft Corporation. All rights reserved.

Contents

[ 1](#_Toc280604197)

[Introduction 4](#_Toc280604198)

[Enabling Partition Writeback on an Analysis Services Database Partition 4](#_Toc280604199)

[UPDATE CUBE Statements 8](#_Toc280604200)

[Commit transaction 10](#_Toc280604201)

[Dynamic Security Using Roles and Factless Fact Tables. 11](#_Toc280604202)

[Assigning Membership to Roles 11](#_Toc280604203)

[Using Factless Fact Tables 11](#_Toc280604204)

[Cleaning Up the Writeback Table 14](#_Toc280604205)

[Using the Update Isolation Level Connection String Property to Improve Writeback Performance 17](#_Toc280604206)

[Conclusion 18](#_Toc280604207)

# Introduction

With the right implementation and in the right scenarios, Analysis Services is transformed from a data aggregation engine to a tool that actually changes the data you are analyzing. This capability is called writeback, and it is the enabling feature behind what-if analysis, forecasting, and financial planning in BI applications that use Analysis Services on the backend.

Why would you want to write data back to Analysis Services rather than the relational database that provides the raw data? One reason is latency. When you write data back to a relational database, users have to wait until the cube is processed before the latest data becomes available in their reports. However, when you enable writeback, users can submit data straight into the cube in the current session, making it instantly visible to other users of the Analysis Services database. This is highly desirable when fast-paced collaboration is the norm, which is often the case in multinational finance organizations where the user base is distributed around the world.

Another reason for using writeback is that the feature has improved significantly over the previous version. One of the drawbacks of the previous version was that the writeback feature relied on ROLAP partitions when writing back data. This meant that every time a user wanted to access the manual writeback partitions data, Analysis Services had to query the relational tables in parallel, which would often result in performance degradation when dealing with a large volume of writeback data. With the introduction of SQL Server 2008, it is now possible to write data back to MOLAP partitions, which enables the same fast data retrieval users have come to expect from Analysis Services. This article will show you how to do the following tasks:

Enable writeback on an Analysis Services database partition.

Set up dynamic security using a factless fact table

Clean up writeback tables using stored procedures

Set connection string properties to improve writeback performance.

This article assumes that the reader is familiar with the basics of SQL Server Analysis Services, MDX, and SQL Server Transact-SQL.

# Enabling Partition Writeback on an Analysis Services Database Partition

In Analysis Services, you can enable writeback on either dimensions or partitions. With write-enabled partitions, a user can modify or supplement the cube data, while dimension writeback provides a mechanism for changing the members of a particular dimension. This article will focus on partition writeback.

When you enable a partition for writeback, values entered by users will be stored in a separate writeback table as a difference from the original value. For example, if a user wants to change a cell from 100 to 150, the writeback table will store a record of +50. If he then wants to change it to 120, the table will have another record of -30. The original cube data (100) remains intact, but at query time Analysis Services will sum the original data with the delta values in the writeback data to produce the current value.

Writeback only works for partitions that contain aggregated data based on SUM, where the cells aggregate to the parent using the SUM function. If the partition includes measures that use different aggregations, you cannot enable writeback.

To enable a partition for writeback follow these steps:

1. In Business Intelligence Development Studio (BIDS), go to the cube designer and then click on the Partitions tab.
2. From the list, right click the desired partition and select **Writeback settings**. The following dialog will appear:



**Table name**: The name of the relational table where Analysis Services will store the writeback data. This table will be automatically created at processing time by default. This behavior can be changed with the *WritebackTableCreation* processing option. The XMLA syntax for it is:

<Batch xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">

 <Parallel>

 <Process xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ddl2="http://schemas.microsoft.com/analysisservices/2003/engine/2" xmlns:ddl2\_2="http://schemas.microsoft.com/analysisservices/2003/engine/2/2" xmlns:ddl100\_100="http://schemas.microsoft.com/analysisservices/2008/engine/100/100">

 <Object>

 <DatabaseID>MYR</DatabaseID>

 </Object>

 <Type>ProcessFull</Type>

 **<WriteBackTableCreation>UseExisting</WriteBackTableCreation>**

 </Process>

 </Parallel>

</Batch>

The possible values are:

**Create**: Create a new writeback table, if one does not exist. If a writeback table already exists, an error occurs. Default value.

**CreateAlways**: Create a new writeback table, overwriting any existing writeback table.

**UseExisting**: Use the existing writeback table, if one already exists. If one does not exist, an error occurs.

The table will have columns for all the measures plus two auditing columns included in the partition and will assign a data type based on Analysis Services data type. However, the names of the columns in the writeback table will be suffixed with a numerical index. For example, given the following table:

 CREATE TABLE [dbo].[FactManual]

(

 [FiscalMonthID] [smallint],

 [Revenue] [numeric](38, 8),

 [Forecast] [numeric](38, 8)

)

The writeback table might look something like this:

 CREATE TABLE [dbo].[WriteTable\_FactManual]

(

[Revenue\_0] [float](38, 8),

[Forecast\_1] [float](38, 8),

[FiscalMonthID\_2] [smallint],

[MS\_AUDIT\_TIME\_3] [datetime],

[MS\_AUDIT\_USER\_4] [nvarchar](255)

)

The order of the columns and suffixes is decided at the time of table creation by Analysis Services and therefore cannot be predicted.

**Data Source**: The data source where the table will be stored. This object points to a relational database that contains the data to be used by the cube. In the context of writeback, it is the relational database where the writeback entries will be stored.

**Storage Mode**: This specifies how the writeback data will be stored in the cube.

The possible values are:

**ROLAP**: The writeback data will be stored only in the relational table. This means when the user issues a query against the partition, Analysis Services will have to retrieve data from the relational partition (as well as cube storage depending on the data the user is querying).

**MOLAP**: The writeback data will be written back to relational as well as in cube storage. The data only gets written to the table once the user issues a successful **COMMIT TRANSACTION** statement (Introduced in SQL Server 2008)

1. After you have entered all of the values in the dialog box, click OK. You should see the new writeback partition in the list.



## UPDATE CUBE Statements

In order to write data back to the cube, a user needs to build an *UPDATE CUBE* statement which has the following syntax:

UPDATE CUBE **Cube\_Name** SET <update clause(s)>

An update clause is made up of a tuple expression followed by the equal sign and a numeric expression representing the desired writeback value.Example:

UPDATE CUBE [Writeback Cube] SET

(

 [Time].[Time].[Fiscal Month].&[436],

 [Measures].[Revenue]

) = 1000

The tuple expression can represent any valid cell in the cube space, including non-leaf members. However, the measures included in the expression cannot be calculated members and must have their Aggregate Function set to SUM.

If the tuple expression is addressing a non-leaf cell, then the writeback engine will perform a series of writeback actions among the cells that make up the sum of the cell specified by the update clause. For example, considering the following UPDATE CUBE statement:

UPDATE CUBE [Writeback Cube] SET

(

 [Time].[Time].[Fiscal Year].&[2011],

 [Geography].[Parent Geography ID].&[7],

 [Measures].[Forecast]

) = 1200

This statement is setting the Forecast measure for the year 2011 in California to a value of 1200. Since time is a user defined hierarchy (Year->Quarter->Month), the value of 1200 will be allocated all the way down to the 12 months that are the leaf level descendants of dimension member year 2011. By default, Analysis Services uses Equal Allocation, which means the value will be split evenly among the 12 members, yielding the following result:



Allocation in parent-child dimensions happens differently. Since parent-child dimensions are allowed to have values not just for the leaf levels, the writeback value will be allocated among the entire branch of the parent-child hierarchy. Example:

UPDATE CUBE [Writeback Cube] SET

(

 [Time].[Time].[Fiscal Month].&[436],

 [Geography].[Parent Geography ID].[United States],

 [Measures].[Forecast]

) = 400

This statement will yield the following result:



The value of 400 split and allocated 100 to each of the 4 members in the branch, including United States, the parent member. Notice how United States is showing a value of 400, which is the sum of the state values (300) plus the value for the US (100).

To override this and write a value directly into a non-leaf member in a parent child dimension, the DATAMEMBER function can be used. Using a statement similar to the one from the previous example, but adding the DATAMEMBER function will yield a different result.

UPDATE CUBE [Writeback Cube] SET

(

 [Time].[Time].[Fiscal Month].&[436],

 [Geography].[Parent Geography ID].[United States].DATAMEMBER,

 [Measures].[Forecast]

) = 400



**Allocation Method**

The allocation method used by Analysis Services may be specified in the UPDATE cube statement at the end of the update clause. Syntax is as follows:

UPDATE CUBE **Cube Name** SET **Tuple Expression** = <**New Value> <Allocation Method> <Weight Expression>**

For more information on allocation methods see the following MSDN article [**http://msdn.microsoft.com/en-us/library/ms145488.aspx**](http://msdn.microsoft.com/en-us/library/ms145488.aspx)

You can specify multiple update clauses separated by commas and execute them in a single statement. When developing a custom application to write back to Analysis Services, this helps reduce network chattiness, as all the updates are sent in a single batch and then committed, as opposed to committing transactions for every update made.

Example:

UPDATE CUBE [Writeback Cube] SET
(

 [Time].[Time].[Fiscal Month].&[436],

 [Measures].[Revenue]

) = 1000 USE\_EQUAL\_ALLOCATION,

(

 [Time].[Time].[Fiscal Month].&[437],

 [Measures].[Revenue]

) = 1500 USE\_EQUAL\_ALLOCATION,

(

 [Time].[Time].[Fiscal Month].&[438],

 [Measures].[Revenue]

) = 2000 USE\_EQUAL\_ALLOCATION

## Commit transaction

After user executes an UPDATE CUBE statement, he or she can see the new data in the cube only for his/her session and active connection. Other users querying the cube from other sessions would still see the unaltered data. At this point, the changes may be reversed by executing a ROLLBACK TRANSACTION statement. To commit the changes and make the data available for everyone accessing the cube, the UPDATE CUBE statement must be followed up by a COMMIT TRANSACTION statement, executed in the same session.

When the COMMIT transaction statement is executed, Analysis Services will issue a SQL query to update the writeback table and after that is completed it will perform an incremental processing of the writeback partition. All of these operations are performed in a single transaction, so if one of the steps fails, then all the changes will be rolled back.

# Dynamic Security Using Roles and Factless Fact Tables.

When developing an application that enables business users to modify or add data, it is very important to have a security model to ensure only users with adequate permissions are able to perform writeback. By leveraging Analysis Services security roles, it is possible to develop a very flexible and robust permission system.

## Assigning Membership to Roles

The first step in enabling write access is to add the user to a database role that has the permissions required for writeback. To create a role with read/write permissions, follow these steps:

1. In BIDS, right click the Roles folder in Solution Explorer and click **New Role.** By default the role will have the “Role” name. Rename if necessary.
2. Click the **Read Definition** and **Process Database** checkboxes in the general tab. **Important:** Process Database is a requisite for a writeback role, since the COMMIT statement performs a process partition operation.
3. Go to the **Cubes** tab and set the **Access** column to “Read/Write” for the desired cubes.
4. In the **Cell Data** tab, select the desired cube on top and then select the **Enable Read/Write permissions** checkbox.In the **Allow reading and writing of cube content** text box, enter an MDX expression that identifies the cells permitted for reading and writing. This expression is evaluated for every cell involved in the UPDATE CUBE statement, and should return a value of true if the cell is allowed to be written to. For examples of MDX statements go [here](http://technet.microsoft.com/en-us/library/ms174590.aspx).

**Note**: If the expression is blank, then the role will have no read/write permissions as the default allowed cell set is an empty set.

1. Add users to the role in the **Membership** tab.

## Using Factless Fact Tables

Using roles to manage security for many users can become complex if your application requires personalized security. For example, every user might have a different set of Geographies he is allowed to write data to, requiring that you create a role for every combination of Geographies. Depending on the number of users, this approach could become very difficult to manage.

A better approach is to store security information for users in relational tables and model it in Analysis Services. The end result is a dynamic security model that requires no modification of cube objects every time user permissions change.

Here are the steps for implementing dynamic write security.

1. Choose a dimension to secure. For this example, we will be securing a Geography dimension, based on the following model:



1. Create a table to store the user information. Make sure you include an Alias column (to store Active Directory user name).

CREATE TABLE [dbo].[Users]

(

 [UserID] [int] NOT NULL,

 [UserAlias] [varchar](50) NOT NULL

)



1. Create the user dimension based on the table created in the last step.
2. Create a table that will hold the relationship between the Geography Dimension.

CREATE TABLE [dbo].[UserGeography]

(

 [UserID] [int] NOT NULL,

 [UserAlias] [varchar](50) NOT NULL,

 [StateID] [int] NOT NULL,

 [WriteFlag] [tinyint] NOT NULL

)



1. Create a measure group from the table in last step. It will look like this:





1. Populate data for the users.

**State**



**Users**



**UserGeography**



1. Use the steps in the section “Setting up write security” to create a writeback role. Add all your users in the Membership tab. In the **Allow reading and writing of cube content** box, enter the following MDX expression:

NOT ISEMPTY(([Geography].[State].CurrentMember, StrToMember("[Users].[User].[" + USERNAME + "]"), [Measures].[Write Flag]))

This expression will check every cell for the current Geography dimension member and also check if there is an entry in the User Geography table for the combination of Geography and Current User.

A different expression may be used and more dimensions can be included.

When a user executes an UPDATE CUBE statement for a cell he or she is not allowed to, the following error will be raised:

Executing the query ...

Errors in cell writeback: Cell writeback has failed because one of the cells is secured.

Execution complete

# Cleaning Up the Writeback Table

*This section is inspired by code found on a blog posting by Boyan Penev, which can be found* [*here*](http://www.bp-msbi.com/2009/01/moving-writeback-data-in-fact-tables/)*.*

Updates made to the cube are stored in a relational table as the difference from the previous value. For example, if a cell has a current value of 100 and an UPDATE cube statement modifies the cell value to 200, the relational table will create a row with a value of 100. Similarly, if a new update is issued changing the same cell to a value of 150, there will now be another row in the table with a value of -50.

Over time, the writeback table will grow larger, negatively impacting performance or storage resources. One approach that we use to mitigate these problems is to create a SQL stored procedure that cleans up the writeback table by aggregating all the delta rows into a single row and moving it to the main fact table.

As mentioned before, the column names on the writeback table and their order are not known until table creation and might change every time the table is recreated. This means that if we hardcode the column names, we will have to modify this procedure every time the table gets recreated. An alternative approach is to create a SQL function that translates the column names from the writeback table so that we don’t have to hardcode them. The code for the function is as follows:

CREATE FUNCTION [dbo].[GetColumnName]

(

 @ColumnName [NVARCHAR](60),

 @WriteBackTableName [NVARCHAR](60)

)

RETURNS [NVARCHAR](60)

AS

BEGIN

 DECLARE @ColumnExactName NVARCHAR(60)

 SET @ColumnExactName = ''

 SELECT @ColumnExactName = Col.[name]

 FROM SYS.COLUMNS Col

 JOIN SYS.TABLES Tbl

 ON Tbl.[object\_id] = Col.[object\_id]

 WHERE Tbl.[name] = @WriteBackTableName

 AND Col.[name] LIKE @ColumnName + '%'

 RETURN(@ColumnExactName)

END

Let’s consider the following two tables for the next examples.

TABLE [dbo].[FactManual](

 [FiscalMonthID] [smallint] NOT NULL,

 [StateID] [int] NOT NULL,

 [Revenue] [numeric](38, 8) NULL,

 [Forecast] [numeric](38, 8) NULL

)

TABLE [dbo].[WriteTable\_Fact Manual](

 [Revenue\_0] [float] NULL,

 [Forecast\_1] [float] NULL,

 [FiscalMonthID\_2] [smallint] NULL,

 [StateID\_3] [int] NULL,

 [MS\_AUDIT\_TIME\_4] [datetime] NULL,

 [MS\_AUDIT\_USER\_5] [nvarchar](255) NULL

)

Calling the function

SELECT dbo.GetColumnName('Revenue', 'WriteTable\_Fact Manual’)

Would result in the following output:

*Revenue\_0*

This function can then be used in a stored procedure to perform the cleanup. Following is a simple example that aggregates the rows in the writeback table and moves them to the main fact table. It could be further enhanced to add up whatever rows are currently present in the main fact table.

SELECT @SQL =

('

 INSERT INTO FactManual

 (FiscalMonthID, StateID, Revenue, Forecast)

 SELECT

 '+ dbo.GetColumnName('FiscalMonthID', 'WriteTable\_Fact Manual) +',

 '+ dbo.GetColumnName('StateID', 'WriteTable\_Fact Manual) +',

 SUM( '+ dbo.GetColumnName('Revenue', 'WriteTable\_Fact Manual) +' ),

 SUM( '+ dbo.GetColumnName('Forecast', 'WriteTable\_Fact Manual) + ')

 FROM dbo.[WriteTable\_Fact Manual]

 GROUP BY FiscalMonthID\_2, StateID\_3

 DELETE FROM dbo.[WriteTable\_Fact Manual]

')

EXEC(@SQL)

# Using the Update Isolation Level Connection String Property to Improve Writeback Performance

There is a connection string property that can help speed up the performance of writeback if your UPDATE CUBE statements consist of multiple update clauses, where none of the cells overlap. If you are not familiar with overlapping cells, the following example illustrates the concept.

UPDATE CUBE [Writeback Cube] SET

(

 [Time].[Time].[Fiscal Month].&[436],

 [Geography].[Hierarchy].[United States],

 [Measures].[Forecast]

) = 1000,

(

 [Geography].[Hierarchy].[United States],

 [Measures].[Forecast]

) = 5000

The first update clause is setting the Forecast to 1000, for fiscal month 436 and the region is United States. In the next update clause, the Forecast is set to 5000 for the United States. Because the second statement doesn’t include the Fiscal Month dimension, the 5000 value will be split across all the members of the dimension, including Fiscal Month 436, which would make it overlap with the previous clause.

After you have verified that your updates do not overlap, you can use the **Update Isolation Level** property to achieve performance gains. A connection string that includes this property looks like this:

Provider=MSOLAP;Data Source=SERVER;Initial Catalog=DB\_NAME;**Update Isolation Level=1;**

The possible values for the property are as follows:

* **Update Isolation Level =1:** Indicates the cells in the update do not overlap.
* **Update Isolation Level =2:** Indicates the cells in the update could overlap. This is the default.

In a small scale test, we tried an UPDATE CUBE statement on 2000 non-overlapping cells. Using the default, where Update Isolation Level=2, we recorded the following times for three consecutive runs:

**UPDATE CUBE: 00:00:06**

**COMMIT TRANSACTION: 00:00:09**

**UPDATE CUBE: 00:00:07**

**COMMIT TRANSACTION: 00:00:10**

**UPDATE CUBE: 00:00:06**

**COMMIT TRANSACTION: 00:00:09**

Running those same tests again, this time with Update Isolation Level=1:

**UPDATE CUBE: 00:00:05**

**COMMIT TRANSACTION: 00:00:08**

**UPDATE CUBE: 00:00:05**

**COMMIT TRANSACTION: 00:00:08**

**UPDATE CUBE: 00:00:04**

**COMMIT TRANSACTION: 00:00:07**

Even in this small scale test, we see a consistent 1 to 2 second improvement in every run. This could translate to even bigger time savings in larger cubes.

# Conclusion

In this article, we shared several proven techniques for enabling, maintaining, and using writeback in an Analysis Services database. We explained how to specify a connection string that results in faster performance for non-overlapping cells, and we covered a common approach used by our IT department to provision dynamic read/write access to writeback data in the database.

For more information, see our companion whitepaper on how to use writeback in Excel, and the following additional resources.

**For more information:**

**Enabling Microsoft Excel to Write Back to an OLAP Cube at Cell Level**

[**http://msdn.microsoft.com/en-us/library/gg521158.aspx**](http://msdn.microsoft.com/en-us/library/gg521158.aspx)

**MSDN - Write-Enabled Partitions**[**http://msdn.microsoft.com/en-us/library/ms174750.aspx**](http://msdn.microsoft.com/en-us/library/ms174750.aspx)

**Granting Cube Access**[**http://technet.microsoft.com/en-us/library/ms174799.aspx**](http://technet.microsoft.com/en-us/library/ms174799.aspx)

**WritebackTableCreation Element (XMLA)**[**http://technet.microsoft.com/en-us/library/ms187172.aspx**](http://technet.microsoft.com/en-us/library/ms187172.aspx)

**Update Cube Statement**[**http://msdn.microsoft.com/en-us/library/ms145488.aspx**](http://msdn.microsoft.com/en-us/library/ms145488.aspx)

**Granting Custom Access to Cell Data**[**http://msdn.microsoft.com/en-us/library/ms174847(v=SQL.100).aspx**](http://msdn.microsoft.com/en-us/library/ms174847%28v%3DSQL.100%29.aspx)

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

* Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?
* Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

Send feedback.