**Auto Generate Your Database Documentation**

**By** [**Andy Jones**](http://www.sqlservercentral.com/Authors/Articles/Andy_Jones/13150/)**, 2016/02/25**

**FROM:** [**http://www.sqlservercentral.com/articles/Extended+Property/137761/**](http://www.sqlservercentral.com/articles/Extended+Property/137761/)

“A society grows great when old men plant trees whose shade they know they shall never sit in.” - Anonymous Greek Proverb.

**Introduction**

Have you ever connected to SQL Server and been overwhelmed by the volume of databases and objects and had no clue as to what each database was for or the purpose of that cryptically named table? As a consultant visiting new client sites, this happens to me all the time and it can take some time to become familiar with a new environment and to establish the purpose of everything. This article concerns documenting everything you add to a database using SQL Server extended properties. This is a fantastic time management technique, allowing you to answer questions about the database design once using a central consistent methodology. It also allows the poor DBA who has to pick up your work a valuable insight when you are no longer around to sit in the shade of the tree.

**Benefits**

Some of the benefits of documenting your database in this way are:

1. **Communicate:** If one developer questions you about database design, it is likely others will require similar clarification. Document what you are doing to communicate clearly with others.
2. **Documentation:** Do you have to document your database for internal/external regulatory purposes? Add extended properties and then auto-generate a professional repeatable document via a third party tool.
3. **Information:** I use the fantastic SQL Prompt from Redgate. These extended properties are exposed via tool tips in SQL Prompt. Hover your mouse in SSMS and get immediate relevant information about the object you are working with - a real time saver.
4. **Portability:** your documentation is within the database itself and is backed up with the database. Your documentation travels with the schema wherever that database is restored.

**Examples**

Some examples of adding extended properties I have found particularly useful are:

**Restored Databases**

If you have cause to restore databases to alternative development/test/staging servers it can be extremely useful to have some information about that database immediately to hand via a tooltip in management studio. I know you don’t restore production databases in such a manner without obfuscating sensitive data, so I won’t even mention that.

USE [master];

GO

DECLARE @LastFullBackup NVARCHAR(100) = N'C:\SQLServerBackup\OFFICE\DBA\FULL\OFFICE\_DBA\_FULL\_20160211\_143456.bak';

DECLARE @DatabaseName sysname = N'DBA';

--back up the tail of the log

BACKUP LOG @DatabaseName TO DISK = N'C:\SQLServerBackup\OFFICE\DBA\LOG\DBA\_LogBackup\_2016-02-11\_14-35-10.bak'

WITH NOFORMAT, NOINIT, NAME = N'DBA\_LogBackup\_2016-02-11\_14-35-10', NOSKIP, NOREWIND, NOUNLOAD, NORECOVERY , STATS = 5;

RESTORE DATABASE @DatabaseName FROM DISK = N'C:\SQLServerBackup\OFFICE\DBA\FULL\OFFICE\_DBA\_FULL\_20160211\_143456.bak'

WITH FILE = 1, NOUNLOAD, STATS = 5, RECOVERY;

USE [DBA];

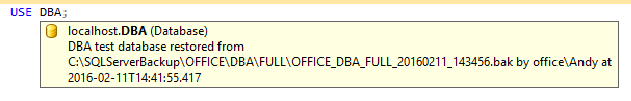
DECLARE @value NVARCHAR(3000) = @DatabaseName

+ ' test database restored from ' + @LastFullBackup + ' by ' + SYSTEM\_USER

+ ' at ' + CONVERT(CHAR(23), CURRENT\_TIMESTAMP, 126);

EXECUTE sys.sp\_addextendedproperty @name = 'MS\_Description', @value = @val

Now, when you hover your mouse in SSMS, you get immediate pertinent information via [SQL Prompt](http://www.red-gate.com/products/sql-development/sql-prompt/):



http://www.sqlservercentral.com/Resources/Images/zoom.gif[Zoom out](javascript:;)  |  [Open in new window](javascript:;)

This immediately tells you who performed the restore using which backup file and when it was done.

**Third party Tools**

A good DBA is continuously learning new techniques and will accumulate knowledge and scripts from others. I was recently doing my long run while training for the 2016 London marathon contemplating the similarities between getting faster at running and performance tuning SQL Server (an article for another day) when I was listening to the compañero Robert Verell on the [SQL Data Partners Podcast](http://sqldatapartners.com/podcast/). His [db\_developer](http://sqlcowbell.com/wordpress/why-nobody-ever-needs-the-db_owner-role/) role is now in my environment and has been credited via:

DECLARE @db\_developer\_extended\_property SQL\_VARIANT;

SET @db\_developer\_extended\_property = SYSTEM\_USER + N' ' + CONVERT(CHAR(23),CURRENT\_TIMESTAMP,126) + N': do not add developers to the db\_owner role as this gives them far too many implicit permissions, add them to the db\_developer role instead - see http://sqlcowbell.com/wordpress/why-nobody-ever-needs-the-db\_owner-role/'

EXECUTE sys.sp\_addextendedproperty @name = 'MS\_Description', -- sysname

@value = @db\_developer\_extended\_property, -- sql\_variant

@level0type = 'USER', -- varchar(128)

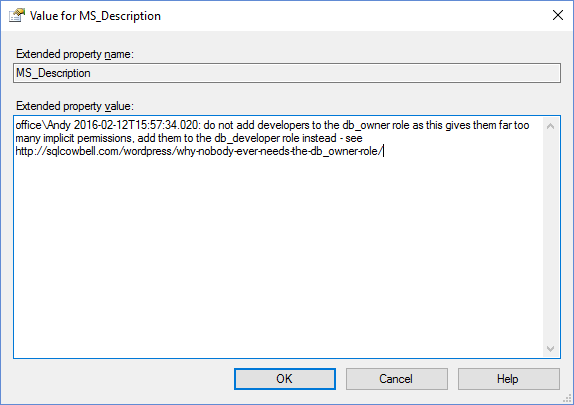
@level0name = 'db\_developer', -- sysname

@level1type = NULL, -- varchar(128)

@level1name = NULL, -- sysname

@level2type = NULL, -- varchar(128)

@level2name = NULL; -- sysname



http://www.sqlservercentral.com/Resources/Images/zoom.gif[Zoom out](javascript:;)  |  [Open in new window](javascript:;)

When the next DBA sees this strange role in their database, they now have full knowledge of why it is there and also a link to investigate for themselves.

One of my favourite sessions while attending PASS 2015 was Steve Wake’s Date Dimension: Past and Future in One Script. Steve’s script to generate the date dimension with all the date attributes you could need already contains extended properties to fully self-document this excellent table. As an aside, I hope Steve is looking forward to the [proposal to fix the date of Easter](https://en.wikipedia.org/wiki/Reform_of_the_date_of_Easter#2014-2016_proposals), which may make his life a little easier.

**Generate missing extended properties**

The following set-based script will generate the statements to create missing extended properties. For brevity, this script demonstrates extended property creation and has been tested for the following object types:

* CHECK\_CONSTRAINT
* COLUMN
* FOREIGN\_KEY\_CONSTRAINT
* PRIMARY\_KEY\_CONSTRAINT
* SQL\_INLINE\_TABLE\_VALUED\_FUNCTION
* SQL\_SCALAR\_FUNCTION
* SQL\_STORED\_PROCEDURE
* UNIQUE\_CONSTRAINT
* USER\_TABLE
* INDEX

This could easily be extended by inserting the relevant 7 rows into @Parameter for any particular object type that supports extended properties.

* This script will not create the extended properties, but auto generate the commands to do so. The actual value (@value parameter) of the extended property still has to be manually input.
* It is possible to automate a description such as “this is column A on table dbo.B” but you will end up with some meaningless documentation that tells the reader very little. I wouldn’t recommend doing this.
* The script will automatically add the user and date as a prefix of the extended property value, remove or edit as desired. This extended property will inform when the object was created and by whom. It is possible to query the database meta-data for this information, but that only tells you the last time it was created. It is possible it has previously been dropped and re-created as part of a change.
* Adds extended properties named MS\_Descripton for automatic inclusion in our third party documentation generator.

/\*

Name:

(C) Andy Jones

mailto:andrew@aejsoftware.co.uk

Example usage: -

Connect SSMS to the database in which you wish to create extended properties and hit F5.

Description: -

This script will not create the extended properties, but auto generate the commands to do so.

The actual value (@value parameter) of the extended property still has to be manually input.

This script will automatically add the user and date to the extended property value.

Change History: -

1.0 25/11/2015 Created.

\*/

DECLARE @Parameter TABLE

(

[type\_desc] sysname NOT NULL ,

parameter sysname NOT NULL ,

val NVARCHAR(100) NOT NULL ,

PRIMARY KEY ( [type\_desc], parameter )

);

/\*set up the data for each object type specifying the correct value for each parameter.\*/

INSERT INTO @Parameter

( [type\_desc], parameter, val )

VALUES ( N'CHECK\_CONSTRAINT', N'value', N'\_replace\_value' ),

( N'CHECK\_CONSTRAINT', N'level0type', N'SCHEMA' ),

( N'CHECK\_CONSTRAINT', N'level0name', N'\_replace\_schemaname' ),

( N'CHECK\_CONSTRAINT', N'level1type', N'TABLE' ),

( N'CHECK\_CONSTRAINT', N'level1name', N'\_replace\_parentname' ),

( N'CHECK\_CONSTRAINT', N'level2type', N'CONSTRAINT' ),

( N'CHECK\_CONSTRAINT', N'level2name', N'\_replace\_name' ),

( N'FOREIGN\_KEY\_CONSTRAINT', N'value', N'\_replace\_value' ),

( N'FOREIGN\_KEY\_CONSTRAINT', N'level0type', N'SCHEMA' ),

( N'FOREIGN\_KEY\_CONSTRAINT', N'level0name', N'\_replace\_schemaname' ),

( N'FOREIGN\_KEY\_CONSTRAINT', N'level1type', N'TABLE' ),

( N'FOREIGN\_KEY\_CONSTRAINT', N'level1name', N'\_replace\_parentname' ),

( N'FOREIGN\_KEY\_CONSTRAINT', N'level2type', N'CONSTRAINT' ),

( N'FOREIGN\_KEY\_CONSTRAINT', N'level2name', N'\_replace\_name' ),

( N'PRIMARY\_KEY\_CONSTRAINT', N'value', N'\_replace\_value' ),

( N'PRIMARY\_KEY\_CONSTRAINT', N'level0type', N'SCHEMA' ),

( N'PRIMARY\_KEY\_CONSTRAINT', N'level0name', N'\_replace\_schemaname' ),

( N'PRIMARY\_KEY\_CONSTRAINT', N'level1type', N'TABLE' ),

( N'PRIMARY\_KEY\_CONSTRAINT', N'level1name', N'\_replace\_parentname' ),

( N'PRIMARY\_KEY\_CONSTRAINT', N'level2type', N'CONSTRAINT' ),

( N'PRIMARY\_KEY\_CONSTRAINT', N'level2name', N'\_replace\_name' ),

( N'UNIQUE\_CONSTRAINT', N'value', N'\_replace\_value' ),

( N'UNIQUE\_CONSTRAINT', N'level0type', N'SCHEMA' ),

( N'UNIQUE\_CONSTRAINT', N'level0name', N'\_replace\_schemaname' ),

( N'UNIQUE\_CONSTRAINT', N'level1type', N'TABLE' ),

( N'UNIQUE\_CONSTRAINT', N'level1name', N'\_replace\_parentname' ),

( N'UNIQUE\_CONSTRAINT', N'level2type', N'CONSTRAINT' ),

( N'UNIQUE\_CONSTRAINT', N'level2name', N'\_replace\_name' ),

( N'SQL\_STORED\_PROCEDURE', N'value', N'\_replace\_value' ),

( N'SQL\_STORED\_PROCEDURE', N'level0type', N'SCHEMA' ),

( N'SQL\_STORED\_PROCEDURE', N'level0name', N'\_replace\_schemaname' ),

( N'SQL\_STORED\_PROCEDURE', N'level1type', N'PROCEDURE' ),

( N'SQL\_STORED\_PROCEDURE', N'level1name', N'\_replace\_name' ),

( N'SQL\_STORED\_PROCEDURE', N'level2type', N'NULL' ),

( N'SQL\_STORED\_PROCEDURE', N'level2name', N'NULL' ),

( N'SQL\_INLINE\_TABLE\_VALUED\_FUNCTION', N'value', N'\_replace\_value' ),

( N'SQL\_INLINE\_TABLE\_VALUED\_FUNCTION', N'level0type', N'SCHEMA' ),

( N'SQL\_INLINE\_TABLE\_VALUED\_FUNCTION', N'level0name',

N'\_replace\_schemaname' ),

( N'SQL\_INLINE\_TABLE\_VALUED\_FUNCTION', N'level1type', N'FUNCTION' ),

( N'SQL\_INLINE\_TABLE\_VALUED\_FUNCTION', N'level1name', N'\_replace\_name' ),

( N'SQL\_INLINE\_TABLE\_VALUED\_FUNCTION', N'level2type', N'NULL' ),

( N'SQL\_INLINE\_TABLE\_VALUED\_FUNCTION', N'level2name', N'NULL' ),

( N'SQL\_SCALAR\_FUNCTION', N'value', N'\_replace\_value' ),

( N'SQL\_SCALAR\_FUNCTION', N'level0type', N'SCHEMA' ),

( N'SQL\_SCALAR\_FUNCTION', N'level0name', N'\_replace\_schemaname' ),

( N'SQL\_SCALAR\_FUNCTION', N'level1type', N'FUNCTION' ),

( N'SQL\_SCALAR\_FUNCTION', N'level1name', N'\_replace\_name' ),

( N'SQL\_SCALAR\_FUNCTION', N'level2type', N'NULL' ),

( N'SQL\_SCALAR\_FUNCTION', N'level2name', N'NULL' ),

( N'USER\_TABLE', N'value', N'\_replace\_value' ),

( N'USER\_TABLE', N'level0type', N'SCHEMA' ),

( N'USER\_TABLE', N'level0name', N'\_replace\_schemaname' ),

( N'USER\_TABLE', N'level1type', N'TABLE' ),

( N'USER\_TABLE', N'level1name', N'\_replace\_name' ),

( N'USER\_TABLE', N'level2type', N'NULL' ),

( N'USER\_TABLE', N'level2name', N'NULL' ),

( N'INDEX', N'value', N'\_replace\_value' ),

( N'INDEX', N'level0type', N'SCHEMA' ),

( N'INDEX', N'level0name', N'\_replace\_schemaname' ),

( N'INDEX', N'level1type', N'TABLE' ),

( N'INDEX', N'level1name', N'\_replace\_parentname' ),

( N'INDEX', N'level2type', N'INDEX' ),

( N'INDEX', N'level2name', N'\_replace\_name' ),

( N'COLUMN', N'value', N'\_replace\_value' ),

( N'COLUMN', N'level0type', N'SCHEMA' ),

( N'COLUMN', N'level0name', N'\_replace\_schemaname' ),

( N'COLUMN', N'level1type', N'TABLE' ),

( N'COLUMN', N'level1name', N'\_replace\_parentname' ),

( N'COLUMN', N'level2type', N'COLUMN' ),

( N'COLUMN', N'level2name', N'\_replace\_name' );

WITH Obj /\*union all objects on which to create extended properties. Objects, columns and indexes.\*/

AS ( SELECT parentname = COALESCE(OBJECT\_NAME(obj.parent\_object\_id),

obj.name) ,

name = obj.name ,

schemaname = SCHEMA\_NAME(obj.[schema\_id]) ,

[type\_desc] = obj.[type\_desc] ,

major\_id = obj.[object\_id] ,

minor\_id = 0,

class\_desc = N'OBJECT\_OR\_COLUMN'

FROM sys.objects AS obj

WHERE obj.is\_ms\_shipped = 0

UNION ALL

SELECT parentname = OBJECT\_NAME(c.[object\_id]) ,

name = c.name ,

schemaname = OBJECT\_SCHEMA\_NAME(c.[object\_id]) ,

[type\_desc] = N'COLUMN' ,

major\_id = c.[object\_id] ,

minor\_id = c.column\_id,

class\_desc = N'OBJECT\_OR\_COLUMN'

FROM sys.columns AS c

WHERE OBJECTPROPERTYEX(c.[object\_id], 'IsMSShipped') = 0

AND OBJECTPROPERTYEX(c.[object\_id], 'IsUserTable') = 1 --only document table columns, not views/functions. Remove predicate if required.

UNION ALL

SELECT parentname = OBJECT\_NAME(i.[object\_id]) ,

name = i.name ,

schemaname = OBJECT\_SCHEMA\_NAME(i.[object\_id]) ,

[type\_desc] = N'INDEX' ,

major\_id = i.[object\_id] ,

minor\_id = i.index\_id,

class\_desc = N'INDEX'

FROM sys.indexes AS i

WHERE OBJECTPROPERTYEX(i.[object\_id], 'IsMSShipped') = 0

AND i.is\_primary\_key = 0 --the constraint is already documented, don't document the index too. Remove predicate if required.

AND i.is\_unique\_constraint = 0 --the constraint is already documented, don't document the index too. Remove predicate if required.

AND i.[type\_desc] <> N'HEAP' --the table is already documented, don't document the heap index row too.

),

/\*Join objects on which to create extended properties to the parameters, performing string replacement where necessary.\*/

Parameter\_Value

AS ( SELECT o.major\_id ,

o.minor\_id ,

o.class\_desc,

p.parameter ,

[name] = N'MS\_Description',

val = CASE p.val

WHEN N'\_replace\_schemaname' THEN o.schemaname

WHEN N'\_replace\_parentname' THEN o.parentname

WHEN N'\_replace\_name' THEN o.name

WHEN N'\_replace\_value'

THEN SYSTEM\_USER + N' '

+ CONVERT(CHAR(10), CURRENT\_TIMESTAMP, 103)

+ N': ' + p.val

ELSE p.val

END

FROM @Parameter AS p

INNER JOIN Obj AS o ON o.[type\_desc] = p.[type\_desc] COLLATE DATABASE\_DEFAULT

)

/\*pivot the result set so we have one correctly formatted extended property create statement per object.\*/

SELECT Add\_Extended\_Property = N'EXECUTE sp\_addextendedproperty'

+ N' @name = ''' + [name] + N''', @value = ' + [value]

+ N', @level0type = ' + [level0type] + N', @level0name = '

+ [level0name] + N', @level1type = ' + [level1type]

+ N', @level1name = ' + [level1name] + N', @level2type = '

+ [level2type] + N', @level2name = ' + [level2name] + N';'

FROM ( SELECT pv.major\_id ,

pv.minor\_id ,

pv.class\_desc,

pv.[name],

pv.parameter ,

val = CASE pv.val

WHEN N'NULL' THEN pv.val

ELSE '''' + pv.val + ''''

END

FROM Parameter\_Value AS pv

WHERE NOT EXISTS ( SELECT \*

FROM sys.extended\_properties AS ep

WHERE ep.major\_id = pv.major\_id

AND ep.minor\_id = pv.minor\_id

AND ep.class\_desc = pv.class\_desc

AND ep.[name] = pv.[name] )

) AS SourceTable PIVOT

( MIN(val) FOR parameter IN ( [value], [level0type], [level0name],

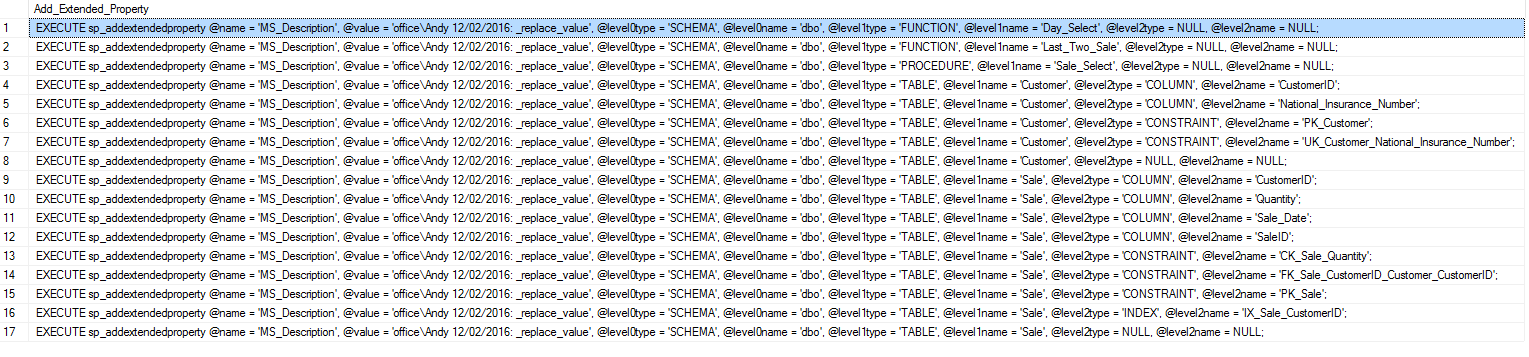
[level1type], [level1name], [level2type],

[level2name] ) ) AS PivotTable

ORDER BY Add\_Extended\_Property;

**Instruction**

First, run the script above against your database. This results in an output such as this:



http://www.sqlservercentral.com/Resources/Images/zoom.gif[Zoom in](javascript:;)  |  [Open in new window](javascript:;)

Next, replace \_replace\_value with your required information. Then run the script to create your extended properties.

**Method to add extended properties to an existing system**

When inheriting a database with no documentation, it can be overwhelming to attempt to document thousands of objects. I generally add extended properties via three methods.

If you get questioned about some part of the database, document what you know via an extended property rather than simply replying via email for example. It is likely others will have the same question in the future and you won’t have to repeat yourself.

For new objects, use snippets or templates. Write the extended property create statement in there and you will always remember to add.

Lastly, pick off existing objects at a few per day or when you have some spare time. Run the script above to generate the statements to create extended properties for currently undocumented objects and create.

**Conclusion**

Some say DBAs are lazy, I would prefer to say efficient. Automate everything that can be automated and focus your skills and productivity at more important and interesting areas.

Don’t answer the same questions from the development team over and over about the database design, clearly and concisely document it once within the database where everyone can see and consume it. [Don't repeat yourself (DRY).](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself)

When the regulatory requirement comes from either within your company or an external body, use a commercial product to auto-generate a beautiful professional looking document that will include your extended properties to describe the function of everything within your database.

And finally, when you have to work on that long forgotten database solution you once delivered because performance is now painfully slow (definitely not your fault, data volumes are now way more than you were told they would be), you can remember why you designed the tables in that way.