**A Standard Database Security Script Generator**

**By** [**Jefferson Elias**](http://www.sqlservercentral.com/Authors/Articles/Jefferson_Elias/1637604/)**, 2015/01/20**

**FROM:** [**http://www.sqlservercentral.com/articles/Generator/120353/**](http://www.sqlservercentral.com/articles/Generator/120353/)

**Introduction**

About two years ago, I had to take care of the SQL Server databases where I work. There was no DBA in charge so I had to start from zero. I've finally designed a standard for SQL Server installations divided in a few sections, in which I include the definition of a Security Model. Here are some guidelines I defined :

* Keep permissions to the strict minimum. Give users only the permissions they need on the objects they use for the way they use them. For instance, do not give INSERT permission on an object the user only needs the SELECT permission.
* Permissions should be assigned at object and schema level, not database or server levels, with exceptions such as CONNECT and CONNECT SQL.
* Permissions should be assigned to database roles so that it's just about membership (with exceptions).
* Every permission assignment needs a reason to be applied. This reason must be documented.
* If possible, system object access (databases like master, msdb, etc.; Schemas like dbo) are not allowed for any user other than DBAs.

I realized a few time afterwards that maintaining such a security model can be very hard, so I decided to be helped by some automation which goals was:

* Produce/generate a SQL script that can be read by anyone and executed on target server.
* The standard can evolve, so we must program it dynamically.
* The way to use it must be easy and crystal clear for another DBA.
* The solution can be centralized or decentralized (local to a server).
* We can keep a history of the generations of the SQL Script applying security defined for a given server.
* Constitute at the same time an inventory and security documentation tool (explain why we gave the permission).

This article is about this automation solution I developed in T-SQL to **document permissions** and **generate a script** according to the security model I designed. The SQL statements that you'll see here are extracted from the solution/application available on [github](https://github.com/jeffchulg/SQLServer-StandardSecurity). Version 1.0.1 is also attached to this article.

Note : this solution is not perfect and doesn't cover everything. It needs to be completed and improved. It can contain bugs in cases I've not tested yet as I'm alone to develop... If you want to participate, don't hesitate to contact me.

**Known issue and limitations**

There is a little shortcut introduced in the implementation at its beginning : a Database User has always the same name as the SQL Login to which it corresponds. As you may see in code, the overall implementation is not affected by this shortcut, but it's considered, at least at the moment, as a requirement. I haven't tested if it works fine in a configuration where the sql login and its corresponding username in a given database are not the same.

Another shortcut is that there is no "InstanceName" we can define for a given server. All the definitions implies then that there is only one instance per server and it should be the default one.

**Workflow of the solution or the way I apply permissions**

The listing below states how I think security should be applied. To me, it's a natural way to define permissions as it goes from instance level to database object-level permissions.

**I. DEFINITION**

1. First, we should define who can connect to the server : so we need to define SQL Logins able to connect to this server.
2. On this server, there are a few databases. For each SQL Login, we should define which databases it can connect and with which default schemas. (We actually create the SQL Login to Database User mapping here)
3. Then we should define which database schemas should be accessed by database users mapped to SQL Logins in step 2.
4. For each database on that server, if we have permissions to assign at schema level, we should define them now. It's a part of my security standard I call "On Schema" role permission assignment. I defined a few database roles that fits to most of my needs (read-only, write-only, ddl-only, etc.).
5. For each database, if we have permissions to assign at object level and they must (or may) be applied to multiple database users, then we should define database roles and assign those users as members of them.
6. For each database, if we have permissions to assign at object level and it's particular to a given user (won't be generalized), we should define them now.

**II. GENERATION**

Now we have defined the security model to apply on a given server, we will be able to apply it once we have run a script generation procedure.

**III. GENERATED T-SQL SCRIPT EXECUTION**

As the script to apply security has been generated, we can now execute it on the server.

***An Important remark :*** There is a little confusion that can come if you pay attention : we define sql logins to database user mappings in Step I.2. In those mappings, we also need to define a default schema for the database user. But, there is no schema definition step before it comes at Step I.3.

Actually, part of Step I.3 is implicitly done during Step I.2 : any database schema referenced during Step I.2 will be automatically defined during this step, so we won't need to spend time for them in Step 1.3.

**Concepts and overview**

This section will cover the concepts defined in this solution and give an overview of what has to be done to complete each step listed in the section above. At each step, if possible, an analogy will be done with what we would do using SQL Server Management Studio.

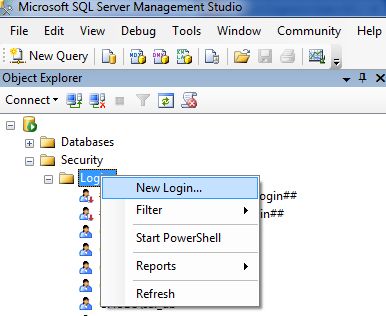
As you may know, the definition of a security model for a given server must be stored permanently. It's achieved by data kept in a set of database tables. There are also some stored procedures that help us to quickly complete a few steps. The objects will also be introduced in each subsection. To get more informations about those database objects, you should go to the implementation section.

**Workflow Step I.1 : Defining who is allowed to connect a given server**

As the solution can be centralized, to define new SQL Logins allowed to connect a SQL Server instance, we must first create an enterprise-level identifier that I call a "Contact". It can be seen as the definition of the person or the application that is behind a login. It's only once a "Contact" has been defined that we can define a SQL Login for a given server.

An example of a "Contact" could be "John Doe", the application manager a given corporate called "MyCorp" which has an active directory login 'MyCorp\john.doe" that we'll use as SQL Login with Windows Authentication.

It's equivalent to the following operation in SSMS :



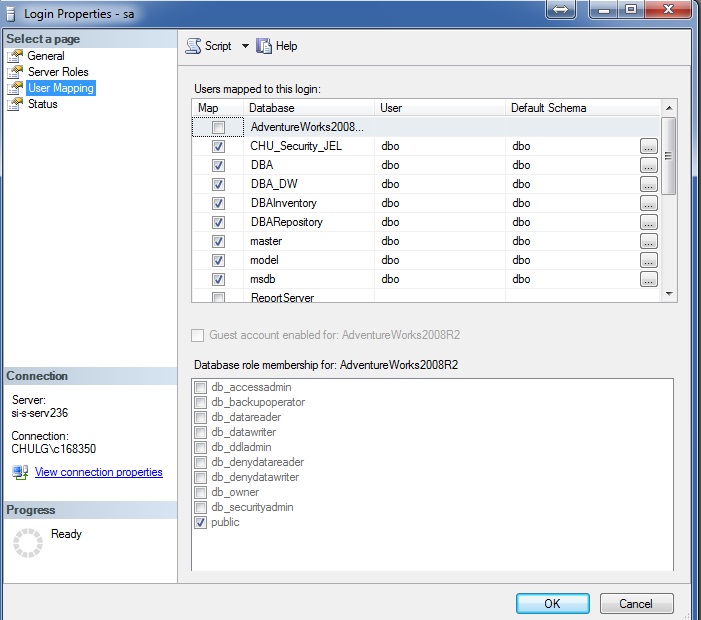
The database table in which contact definitions are stored is **[security].[Contacts]** and the database table in which login definitions are stored is **[security].[SQLLogins]**.

This step can be completed by directly inserting into those two tables. The SQL Logins definition can be done by using a stored procedure called **[security].[setServerAccess]**.

We'll discuss about those objects in details in the section about the implementation of the solution.

**Workflow's step I.2 : Defining Login to User Mappings**

This is what we will refer to as "SQL Mappings". It's similar to what we do when we get to this view and select checkboxes in the pane titled "Users mapped to this login".



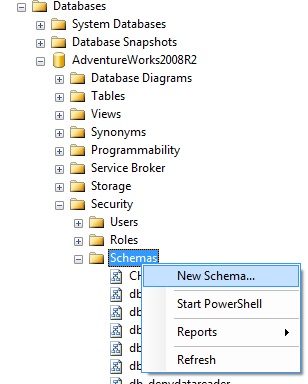
http://www.sqlservercentral.com/Resources/Images/zoom.gif[Zoom in](javascript:;)  |  [Open in new window](javascript:;)

We define the SQL mappings with the same informations (and sometimes more). The database table in which SQL mappings definitions are stored is **[security].[SQLMappings].**We can complete this step either by directly inserting in this table or by using a stored procedure called **[security].[setDatabaseAccess]**.

We'll discuss about those objects in details in the section about the implementation of the solution.

**Workflow's step I.3 : Defining schemas that has to exist in each database**

Defining the schemas is similar to creating new database schemas in this way using SSMS:



As explained above, a part of this step is implicitly performed during the step I.2 in that to complete step I.2, we must assign a default database schema to each database user for  which we map to a given SQL Login for a given database. If those default schemas aren't defined in our solution, step I.2 will create them for us.

To complete this step, we must insert into a table named **[security].[DatabaseSchemas]**. We will show you an example of this behaviour in the example usage section.

As with every object until now, this table will be covered in the implementation section below.

**Workflow's step I.4 : Defining permissions at schema level**

SQL Server, there are no roles defined for assigning permissions on a given database schema. It's one of the things I bring with my solution : I defined a set of roles in my security standard to allow access for all the objects in a given schema. This is what I call the "Standard On Schema" roles. Let's have a look at each of them.

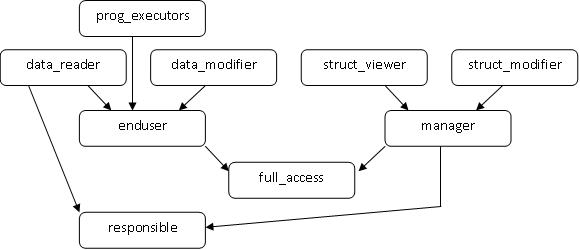
**Concept : Standard On Schema Roles definition**

There are a few roles provided by default in my solution. They are listed below:

|  |  |
| --- | --- |
| **Solutions's standard "On Schema" roles** | |
| data\_reader | has SELECT permission on given schema |
| data\_writer | has INSERT,UPDATE,DELETE permissions on given schema |
| prog\_executors | has EXECUTE on schema |
| endusers | can run an application against the given schema |
| struct\_viewer | can view schema object definitions |
| struct\_modifier | can create, modify and drop objects in the given schema |
| manager | is nothing more that member of roles struct\_modifier and struct\_viewer |
| Responsible | is 'manager' and 'data\_reader' |
| Full\_access | can do anything in the given schema |

To avoid collisions in a database, these roles will be generated as **<SCHEMA\_NAME>\_<ROLE\_NAME>** by default. I say "by default", because the separator used (underscore) is customizable.

As you may have guessed, these roles have dependencies. The figure below shows the hierarchy between them :



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

In terms of storage, Standard On Schema role names are stored in the table called **[security].[StandardOnSchemaRoles]**. Their corresponding permissions and role assignments are stored the table called **[security].[StandardOnSchemaRolesSecurity]**.

Despite the fact that we have a standard, there may be cases where this standard cannot be applied. For instance, there is no possibility to apply our security model for the [INFORMATION\_SCHEMA] database schema. That's the reason why we have a table called **[security].[StandardExclusion]**.

We will cover those tables in detail in the implementation section.

**Standard roles generation**

To complete this step, we need to first generate standard "On Schema" roles. We will use the procedure called **[security].[setStandardOnSchemaRoles]** to do so. We'll cover it in details in the implementation section, but for now, let's say that this procedure will generate database roles using the naming convention defined above (**<SCHEMA\_NAME>\_<ROLE\_NAME>**) and using the three database tables introduced in the previous subsection. The roles will be inserted in a table named **[security].[DatabaseRoles]**.

In this step, the solution will also copy the object permissions and role assignments defined in **[security].[StandardDatabaseRolesSecurity]**. The object permissions will be stored in a table called **[security].[DatabasePermissions]** and the role assignments in a table called **[security].[DatabaseRoleMembers]**.

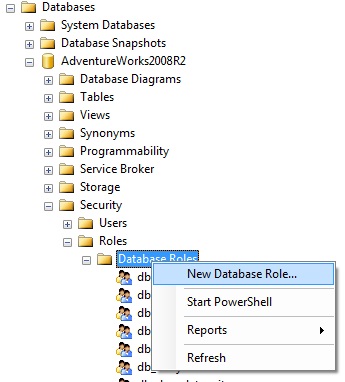
For further information, have a look at implementation section.

**Standard roles assignments**

Once we defined which Standard On Schema database role has to be applied for each database user, we will use a procedure called **[security].[setStandardOnSchemaRole]** to set this role for each database user. This stored procedure will actually assign the given user to a standard role defined in **[security].[DatabaseRoles]** by the procedure **[security].[setStandardOnSchemaRoles]**. This will result in the addition of rows in the table called **[security].[DatabaseRoleMembers]**.

**Workflow's step I.5 : Assign database roles that are outside our standard**

This step is similar to creating a new database role in SSMS :



In our solution, we complete this step by doing the following for each database role we need to create :

1. Define a new Database role by inserting a row in **[security].[DatabaseRoles].**
2. Define permissions for this role by inserting in **[security].[DatabasePermissions].**
3. Define which database roles this new role inherits by inserting in **[security].[DatabaseRoleMembers]**.
4. Define members of this new role by inserting in **[security].[DatabaseRoleMembers].**

**Remark / Definition**

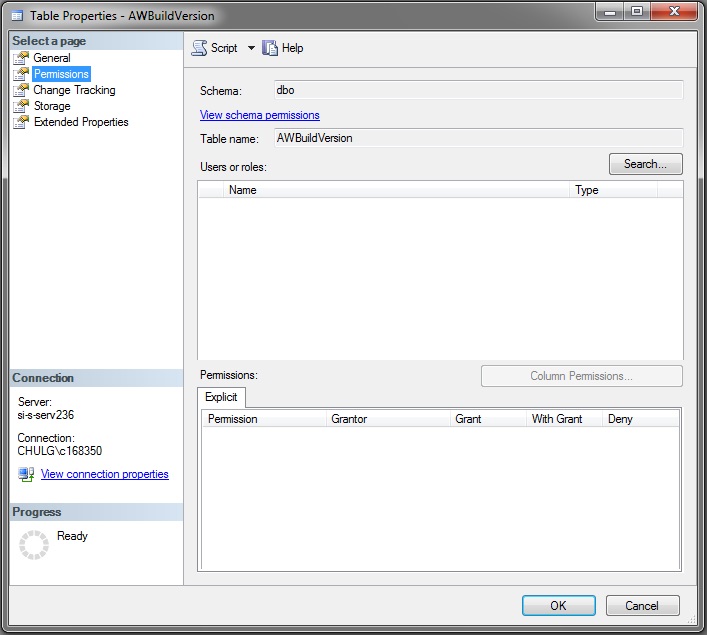
Maybe you noticed that these are the same kind of operations that are performed for Standard On Schema security permissions assignments, except that this operation is simplified by the use of stored procedures, as this is a repetitive task.

Let's introduce a new concept: the concept of "helpers". A stored procedure that allows us to perform operations faster against a set of database tables in our security solution is called a "helper".

**Workflow's step I.6 : assigning object permissions**

If you have followed previous subsections, you may have understood that those permissions are defined by inserting in a table called **[security].[DatabasePermissions]**.

The equivalent operation can be done in SSMS in this panel :



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

**Workflow's step II : Generate a T-SQL script for applying security**

As explained in the introduction, all we've done so far is just ***defining*** a security model for a given server or set of servers. It's now time to do something that will give us the ability to ***act*** or set things on that server or set of servers.

To generate a T-SQL script based up on the definitions we made in step I, we will use a stored procedure called **[security].[getSecurityScript]**. This procedure will print out on standard output. An example of generated script is attached to this article.

**Workflow's step III : Execute the generated T-SQL script**

All we need to do is to take the generated script and execute it against the server.

**Note:** I plan on creating a script, in Powershell or Perl, that will take care of steps II and III.

**How to install the solution**

There is a file named "**securityApplier\_SQLServer\_1.0.1**.sql" attached to this article. It's also available in the "dist" directory on [github](https://github.com/jeffchulg/SQLServer-StandardSecurity" \t "_blank). It contains all the instructions to build the solutioin. It creates a database schema named "[Security]" and the objects of our solution.

Here are the steps to install :

1. Create a new database on the server of your choice
2. In SSMS, open a new query against the server with current database set to the newly created database
3. Copy-Paste the content of this file
4. Run the script

If you experience any issue during the installation, please report the error on github.

**A word about the implementation**

We will now discuss about the database objects that are used in our solution. We'll start by listing database tables and a little description. There are some tables that have data just after installation. We'll talk about each of them.

**Database tables**

If you have read the section introducing concepts and giving you an overview of our solution, you may have seen a lot of database tables, all in a schema named [security]. They are all listed in the table below with a little description.

| Inventory object tables | |
| --- | --- |
| **Table Name** | **Description** |
| Contacts | will answer the question              "Who's login is it?" |
| SQLLogins | will answer the question              "On which servers can a contact log in ?" |
| SQLMappings | will answer the question              "On which databases can can a contact log in ?" |
| DatabaseSchemas | will list all the database schemas for a given database on a given server |
| DatabaseRoles | will list all the database roles for a given database on a given server |
| DatabaseRoleMembers | will answer the question              "Who is what on this server ?" |
| DatabasePermissions | will list all the permissions assigned to any database user or database role that are listed in DatabaseRoles and SQLLogins |
|  |  |
| StandardOnSchemaRoles | This table will store the standard roles definition (data\_reader,full\_access, and so on).  This table has default data after installation. |
| StandardOnSchemaRolesSecurity | This table will store the permissions assigned  to a given standard role.  This table has default data after installation. |
| StandardExclusion | This table will be used to define exceptions to the defined standard. For instance, we won't be able to apply this security standard for the INFORMATION\_SCHEMA schema.  This table has default data after installation. |
|  |  |
| ApplicationLog | This table will be a central execution log for any program or procedure that we want.  It's not used at the moment, but should be used in the future. |
| ApplicationParams | This table will store every parameter needed by the solution.  This table has default data after installation. |

**Table StandardOnSchemaRoles**

If you want, you can run the following query to get the content of this table :

select \* from Security.StandardOnSchemaRoles

and you'll get the following result :

RoleName Description isActive CreationDate lastmodified

--------------------- ----------- -------- ----------------------- -----------------------

data\_modifier NULL 1 2014-04-23 00:00:00.000 2014-12-24 14:21:51.563

data\_reader NULL 1 2014-04-23 00:00:00.000 2014-12-24 14:21:51.563

endusers NULL 1 2014-04-23 00:00:00.000 2014-12-24 14:21:51.563

full\_access NULL 1 2014-04-23 00:00:00.000 2014-12-24 14:21:51.563

managers NULL 1 2014-04-23 00:00:00.000 2014-12-24 14:21:51.563

prog\_executors NULL 1 2014-11-25 00:00:00.000 2014-12-24 14:21:51.563

responsible NULL 1 2014-04-23 00:00:00.000 2014-12-24 14:21:51.563

struct\_modifier NULL 1 2014-04-23 00:00:00.000 2014-12-24 14:21:51.563

struct\_viewer NULL 1 2014-04-23 00:00:00.000 2014-12-24 14:21:51.563

You may notice that they are the roles we discussed above as our standard roles for "on schema" permission assignment. The isActive column is not taken into account at the moment but may be used in the future to set a role as part of the standard or not.

**Table StandardOnSchemaRolesSecurity**

If you want, you can run the following query to get the content of this table :

select \* from Security.StandardOnSchemaRolesSecurity -- where RoleName = 'data\_modifier'

Here is the output for the standard on schema roles named "data\_modifier" and "endusers" :

RoleName PrivName isRoleMembership PermissionClass PermissionLevel isActive CreationDate lastmodified

------------- -------- ---------------- --------------- --------------- -------- ----------------------- -----------------------

data\_modifier DELETE 0 DATABASE\_SCHEMA GRANT 1 2014-12-24 14:21:52.617 2014-12-24 14:21:52.623

data\_modifier EXECUTE 0 DATABASE\_SCHEMA REVOKE 1 2014-12-24 14:21:52.617 2014-12-24 14:21:52.623

data\_modifier INSERT 0 DATABASE\_SCHEMA GRANT 1 2014-12-24 14:21:52.617 2014-12-24 14:21:52.623

data\_modifier UPDATE 0 DATABASE\_SCHEMA GRANT 1 2014-12-24 14:21:52.617 2014-12-24 14:21:52.623

endusers data\_modifier 1 DATABASE\_SCHEMA GRANT 1 2014-12-24 14:21:52.617 2014-12-24 14:21:52.623

endusers data\_reader 1 DATABASE\_SCHEMA GRANT 1 2014-12-24 14:21:52.617 2014-12-24 14:21:52.623

endusers prog\_executors 1 DATABASE\_SCHEMA GRANT 1 2014-12-24 14:21:52.617 2014-12-24 14:21:52.623

It's the permissions we talked about when introducing standard on schema role data\_modifier.

The isRoleMembership is set to 1 if the PrivName column is filled with the name of another standard on schema role as it's the case for "endusers" role.

The PermissionClass column can have the following values at the moment : "DATABASE\_SCHEMA" or "DATABASE". It will be renamed in the future to StandardRolesSecurity as we plan to allow create standard roles for database permissions.

The PermissionLevel column can have the following values : 'GRANT','REVOKE' or 'DENY'.

The isActive column is not taken into account at the moment but may be used in the future to set a permission as part of the standard or not.

**Table StandardExclusion**

At the moment, the exclusions/restrictions listed in this table only have effect on the generation of "on Schema roles".

If you execute the following query :

select ObjectType,ObjectName,Description,isActive from Security.StandardExclusion

you'll get the content of StandardExclusion table after installation listed below :

ObjectType ObjectName Description isActive

--------------- ------------------- ------------------------------------------------------------------------------------------------------------ --------

DATABASE master System database - should be used wisely and for punctual permission assignments 1

DATABASE model System database - is of no use as itself for permissions. Way better to make it for each database created ! 1

DATABASE msdb System Utility database - should be used wisely and for punctual permission assignments 1

DATABASE tempdb System Utility database - no known usage of implementing security in this database 1

DATABASE\_SCHEMA db\_accessadmin System schema - should be used wisely and for punctual permission assignments 1

DATABASE\_SCHEMA db\_backupoperator System schema - should be used wisely and for punctual permission assignments 1

DATABASE\_SCHEMA db\_datareader System schema - should be used wisely and for punctual permission assignments 1

DATABASE\_SCHEMA db\_datawriter System schema - should be used wisely and for punctual permission assignments 1

DATABASE\_SCHEMA db\_ddladmin System schema - should be used wisely and for punctual permission assignments 1

DATABASE\_SCHEMA db\_denydatareader System schema - should be used wisely and for punctual permission assignments 1

DATABASE\_SCHEMA db\_denydatawriter System schema - should be used wisely and for punctual permission assignments 1

DATABASE\_SCHEMA db\_owner System schema - should be used wisely and for punctual permission assignments 1

DATABASE\_SCHEMA db\_securityadmin System schema - should be used wisely and for punctual permission assignments 1

DATABASE\_SCHEMA dbo System schema - should be used wisely and for punctual permission assignments 1

DATABASE\_SCHEMA guest System schema - should be used wisely and for punctual permission assignments 1

DATABASE\_SCHEMA information\_schema System schema - should be used wisely and for punctual permission assignments 1

DATABASE\_SCHEMA sys System schema - should be used wisely and for punctual permission assignments 1

By default, there are two restrictions :

1. We don't generate "on schema" roles in system databases. (Where objectType column is "DATABASE")
2. We don't generate "on schema" roles for database schemas created by default by SQL Server (where ObjectType column is "DATABASE\_SCHEMA"). There can be cases where one has user objects in "dbo". In that case, he should remove this line. The "isActive" column will be taken into account in a fewer version.

**Table ApplicationParams**

The following query will list all the parameters created at installation.

select \* from [Security].ApplicationParams

There are the following parameters :

* MSSQL\_LoginSecurity\_DefaultPassword  : will store the default password to use when creating a new SQL Login
* ObjectPermissionGrantorDenier               : will store the default grantor to use when applying permissions (Cf. [Microsoft GRANT syntax](http://msdn.microsoft.com/en-us/library/ms187965.aspx), it's the [ AS principal ] clause)
* RoleAuthorization4Creation                     : same as ObjectPermissionGrantorDenier, but for role creation
* SchemaAuthorization4Creation                : same as RoleAuthorization4Creation, but for schema creation
* Separator4OnSchemaStandardRole         : to set the separator between <SCHEMA\_NAME> and <STANDARD\_ROLE\_NAME> in standard "on schema" name generation. By default, it's '\_'.
* Version                                                 : stores the version name for the solution. It will help for version upgrades.

**Set of Definition procedures and helpers**

Here is a presentation of a set of procedures I like to refer to as "helpers", as they really help us in the security definition process.

**setStandardOnSchemaRoles - Procedure which defines standard roles and their corresponding permissions**

With this procedure, there is no need to create standard on schema roles by yourself. Based on its parameters, it defines a set of standard roles that have to be created and add them to the DatabaseRoles table. It also adds permissions in DatabasePermissions table for those roles according to their definition in StandardOnSchemaRolesSecurity table.

It can be run at different levels :

* overall server
* overall database on a given server
* all roles relative to a given schema in a given database on a given server
* a given standard role defined in StandardOnSchemaRoles table for a given schema in a given database on a given server

**Procedure definition**

[security].[setStandardOnSchemaRoles] (

@ServerName varchar(512) = @@SERVERNAME,

@DbName varchar(64) = NULL,

@SchemaName varchar(64) = NULL,

@StdRoleName varchar(64) = NULL,

@Debug BIT = 0

)

**Example usage (full server settings)**

exec [security].[setStandardOnSchemaRoles] @ServerName = 'MyServer1'

**Example usage (given database)**

exec [security].[setStandardOnSchemaRoles] @ServerName = 'MyServer1', @DbName = 'ApplicationDatabase1'

**Example usage (given schema)**

exec [security].[setStandardOnSchemaRoles]

@ServerName = 'MyServer1',

@DbName = 'ApplicationDatabase1',

@SchemaName = 'ApplicationSchema1'

**What have changed after execution ?**

With the following query, you'll get an overview of the database roles that have been defined by this procedure in DatabaseRoles table.

select \*

from security.DatabaseRoles

where ServerName = 'MyServer1'

Here is an example of what you get if you run the example usage entitled "Example usage (given schema)" above.

ServerName DbName RoleName isStandard Reason isActive

---------- ------------------- -------------------------------- ---------- ------ -------

MyServer1 ApplicationDatabase1 ApplicationSchema1\_data\_modifier 1 NULL 1

MyServer1 ApplicationDatabase1 ApplicationSchema1\_data\_reader 1 NULL 1

MyServer1 ApplicationDatabase1 ApplicationSchema1\_endusers 1 NULL 1

MyServer1 ApplicationDatabase1 ApplicationSchema1\_full\_access 1 NULL 1

MyServer1 ApplicationDatabase1 ApplicationSchema1\_managers 1 NULL 1

MyServer1 ApplicationDatabase1 ApplicationSchema1\_prog\_executors 1 NULL 1

MyServer1 ApplicationDatabase1 ApplicationSchema1\_responsible 1 NULL 1

MyServer1 ApplicationDatabase1 ApplicationSchema1\_struct\_modifier 1 NULL 1

MyServer1 ApplicationDatabase1 ApplicationSchema1\_struct\_viewer 1 NULL 1

You can also go and check the DatabasePermissions table by adapting the following query :

select ServerName, DbName, Grantee, isUser, ObjectClass, ObjectType,

PermissionLevel, PermissionName, SchemaName, ObjectName

from security.DatabasePermissions

where ServerName = 'MyServer1'

and DbName = 'ApplicationDatabase1'

and Grantee = 'ApplicationSchema1\_prog\_executors'

And here are its results

ServerName DbName Grantee isUser ObjectClass ObjectType PermissionLevel PermissionName SchemaName ObjectName

---------- -------------------- --------------------------------- ------ --------------- ---------- --------------- -------------- ---------- ------------------

MyServer1 ApplicationDatabase1 ApplicationSchema1\_prog\_executors 0 DATABASE\_SCHEMA NULL GRANT EXECUTE NULL ApplicationSchema1

So here, the database role ApplicationSchema1\_prog\_executors in database named ApplicationDatabase1 on server MyServer1 has been granted the EXECUTE permission on database schema ApplicationSchema1.

**setServerAccess - Helper to allow a set of contacts to connect to a Server**

This procedure will generate a list of SQL logins based on

* Contacts table contents
* At least one of the filter parameters either on Department, Job Title or contact Name. It can be either a "=" or "LIKE" operation with the parameter @exactMatch

Example Usage :

DECLARE @ServerName VARCHAR(512) = 'MyServer1'

Exec [security].[setServerAccess]

@ServerName = @ServerName,

@@ContactLogin = NULL,

@ContactDepartment = 'MyCorp/IT%',

@ContactsJob = NULL,

@ContactName = NULL,

@exactMatch = 0

So, if we have two contacts defined in Contacts table, which are members of the IT department, then their login will be added to the SQLLogins table.

If you want to check what it did, you can run the following query :

select \* from security.Contacts where Department like 'MyCorp/IT%'

select \* from security.SQLLogins where ServerName = 'MyServer1'

**setDatabaseAccess - Helper to allow a set of contacts to connect to a database**

This procedure will generate a list of SQL logins as "setServerAccess" does and add records in SQLMappings table. By the way, if the parameter @withServerAccessCreation is set to 1, this procedure will call setServerAccess to allow the access. If not, it will check for potential logins and only perform operations for the logins that are defined in SQLLogins table.

Example Usage :

DECLARE @ServerName VARCHAR(512) = 'MyServer1'

Exec [security].[setDatabaseAccess]

@ServerName = @ServerName,

@DbName = 'ApplicationDatabase1',

@DefaultSchema = 'ApplicationSchema1',

@isDefaultDb = 1,

@withServerAccessCreation = 1,

@ContactLogin = null ,

@ContactDepartment = 'MyCorp/IT%',

@ContactsJob = NULL,

@ContactName = NULL,

@exactMatch = 0

So, if we have 2 contacts defined in Contacts table which are members of the IT department, then

1. As @WithServerAccessCreation is set to 1, their login will be added to the SQLLogins table.
2. One record per login will be created in SQLMappings

If you want to check what it did, you can run the following query :

select \* from security.Contacts where Department like 'MyCorp/IT%'

select \* from security.SQLLogins where ServerName = 'MyServer1'

select \*

from security.SQLMappings

where ServerName = 'MyServer1'

and DbName = 'ApplicationDatabase1'

WARNING : There is a limitation introduced by this function : you may have noticed that there is no @DbUserName parameter. There is a shortcut in the implementation of this procedure which is that the SQLLogin created and the database user to which it is mapped will be the same.

**Set of Generation procedures**

If you are interested in the overall generation algorithm, I invite you to go on [github](https://github.com/jeffchulg/SQLServer-StandardSecurity" \t "_blank), in src/sqlserver and open the file named

* Procedure.getSecurityScript.sql

It's the procedure that will be called to generate a security script for either a given server or a given database on that server. A top-overview of the algorithm is as follows :

"Generate statements for servername check"

"Generate statements to create/alter/drop logins if necessary"

foreach db in databases {

"Generate statements to check database exists"

"Generate statements to create/alter/drop database schemas if necessary"

"Generate statements to create/alter/drop database users if necessary"

"Generate statements to set/remove mappings between sql logins and database users, if necessary"

"Generate statements to create/alter/drop database roles if necessary"

"Generate statements to add/remove database role membership, if necessary"

"Generate statements to grant/revoke/deny permissions to sql logins/database users/database roles"

}

There is an example of generated script attached to this article. It's related to the scenario described in next section.

**Example usage : security definition for fresh server install**

Let's say we have a new server in our company, and this is the first time we use the solution. The server will have a database named ApplicationDatabase1 in which there are 2 database schemas ApplicationSchema1 and ApplicationSchema2.

We'll first create some contacts (Organization-wide logins which is defined at least as a SQL Login) to define who is who.

* John Doe is an application manager in our company defined in our active directory. His login is MyDomain\JohnDoe.
* I'm the DBA (Jefferson Elias) of the company defined in our active directory. My login is MyDomain\Jeff.Elias
* There is an end-user SQL Login "enduser1" to create

Then we'll set that those contacts can connect on the new server and which database they can use. Afterwards, we'll define the database schemas.

Once every top-level element (server, databases, database schemas) is defined, we can launch the procedure responsible for standard security definition on them. Then we'll do some customization for convenience (custom roles definition, custom permission assignment and so on). Finally we can generate a script that will allow us to apply the defined security on our server. This script will create sql logins, database users, database schemas, database roles if they doesn't exist on the given database server.

**Create contacts**

We'll first create some contacts

-- Create new IT contact

insert into [security].[Contacts]

(SqlLogin,Name,job,isActive,Department,authmode)

values (

'MyDomain\JohnDoe',

'John Doe',

'Application Manager',

1,

'MyCorp/IT Service/Application Support Team',

'WINDOWS'

);

-- Create new DBA

insert into [security].[Contacts]

(SqlLogin,Name,job,isActive,Department,authmode)

values (

'MyDomain\Jeff.Elias',

'Jefferson Elias',

'Oracle and SQL Server DBA',

1,

'MyCorp/IT Service/DBA Team',

'WINDOWS'

);

-- Create new end user

insert into [security].[Contacts]

(SqlLogin,Name,job,isActive,Department,authmode)

values (

'Enduser1',

'Application End User 1',

'N/A',

1,

'External/DevCorp/DevProduct',

'SQLSRVR'

);

**Define logins for a given server**

There are two possible options to perform this operation. Let's have a look at both.

**Option 1 : By inserting in table SQLLogins.**

Here we don't add a row in any "Servers" table. Maybe we should, but it's not done at the moment...

insert into security.SQLlogins (

ServerName,

SqlLogin,

isActive

)

values (

'MyServer1',

'MyDomain\JohnDoe',

1

)

insert into security.SQLlogins (

ServerName,

SqlLogin,

isActive

)

values (

'MyServer1',

'MyDomain\Jeff.Elias',

1

)

insert into security.SQLlogins (

ServerName,

SqlLogin,

isActive

)

values (

'MyServer1',

'Enduser1',

1

)

**Option 2 : By using setServerAccess stored procedure**

It can be done by the following queries :

exec [security].[setServerAccess] @ServerName = 'MyServer1', @ContactLogin = 'MyDomain\JohnDoe'

exec [security].[setServerAccess] @ServerName = 'MyServer1', @ContactLogin = 'MyDomain\Jeff.Elias'

exec [security].[setServerAccess] @ServerName = 'MyServer1', @ContactLogin = 'Enduser1'

**Login to database user mappings**

There are also two possible options to perform this operation. Let's have a look at both.

**Option 1 : By inserting into [SQLMappings] table**

Let's now create the sql mappings for our user to be able to connect to databases

insert into security.SQLMappings (

ServerName,

SqlLogin,

DbName,

DbUserName,

DefaultSchema,

isDefaultDb,

isLocked

)

values (

'MyServer1',

'MyDomain\JohnDoe',

'ApplicationDatabase1',

'MyDomain\JohnDoe',

'ApplicationSchema1',

1,

0

)

insert into security.SQLMappings (

ServerName,

SqlLogin,

DbName,

DbUserName,

DefaultSchema,

isDefaultDb,

isLocked

)

values (

'MyServer1',

'MyDomain\Jeff.Elias',

'master',

'MyDomain\Jeff.Elias',

'dbo',

1,

0

)

insert into security.SQLMappings (

ServerName,

SqlLogin,

DbName,

DbUserName,

DefaultSchema,

isDefaultDb,

isLocked

)

values (

'MyServer1',

'Enduser1',

'ApplicationDatabase1',

'Enduser1',

'ApplicationSchema1',

1,

0

)

**Option 2 : By using setDatabaseAccess stored procedure**

Below are the commands with the same effect as above. Notice that there is a parameter named @withServerAccessCreation . If this parameter was set to 1, then there was no obligation to create SQL Logins before calling this stored procedure.

exec setDatabaseAccess

@ServerName = 'MyServer1',

@DbName = 'ApplicationDatabase1',

@ContactLogin = 'MyDomain\JohnDoe',

@DefaultSchema = 'ApplicationSchema1',

@isDefaultDb = 1,

@withServerAccessCreation = 0

exec setDatabaseAccess

@ServerName = 'MyServer1',

@DbName = 'master',

@ContactLogin = 'MyDomain\Jeff.Elias',

@DefaultSchema = 'dbo',

@isDefaultDb = 1,

@withServerAccessCreation = 0

exec setDatabaseAccess

@ServerName = 'MyServer1',

@DbName = 'ApplicationDatabase1',

@ContactLogin = 'Enduser1',

@DefaultSchema = 'ApplicationSchema1',

@isDefaultDb = 1,

@withServerAccessCreation = 0

**Schema definitions**

We'll now get into creating the database schemas that we'll use. At this point, we used a little shortcut to database schema definition : there are records in table DatabaseSchemas even though we didn't touch it !

select \* from security.DatabaseSchemas

| **ServerName** | **DbName** | **SchemaName** | **Description** | **isActive** | **CreationDate** | **LastModified** |
| --- | --- | --- | --- | --- | --- | --- |
| MyServer1 | ApplicationDatabase1 | ApplicationSchema1 | Created automatically from SQLMappings table | 1 | 2014-12-23 13:27:01.327 | 2014-12-23 13:27:01.333 |
| MyServer1 | master | dbo | Created automatically from SQLMappings table | 1 | 2014-12-23 13:27:01.347 | 2014-12-23 13:27:01.347 |

If we are done, good, we just spare some time, Just make sure the description field fits to your needs as it's not really relievant. If not, here is how to add an additionnal schema :

insert into security.DatabaseSchemas (

[ServerName],

[DbName],

[SchemaName],

[Description],

[isActive]

)

values (

'MyServer1',

'ApplicationDatabase1',

'ApplicationSchema2',

'Another application schema, for application management',

1

)**.**

**Auto-creation of standard roles**

Now we have set all the roles we have, we can get into security documentation. Firstly, we'll use the following procedure to generate automatically all the roles according to the defined standard.

exec [security].[setStandardOnSchemaRoles] @ServerName = 'MyServer1'

The result can be seen with the following queries :

-- you'll see all the standard roles prefixed by the schema name

select \* from security.DatabaseRoles

-- you'll see the standard role membership

select \* from security.DatabaseRoleMembers

-- you'll see the permissions defined with the standard

select \* from security.DatabasePermissions

**Our job : Custom roles, Role membership and permission assignment**

We'll make some settings to make everything work :

1. As we have two database schemas and at least one enduser, we'll create a custom role "ApplicationEndUsers" which will be member of both <schema\_name>\_endusers database roles.
2. Then we'll set the enduser as member of this new role.
3. We'll do the same with a role "ApplicationResponsibles"
4. I generally log every DDL operation in a table called "dbo.SchemaChangeLog". At my job, the application managers are responsible for schema object creation. So they must have INSERT privilege for this table. It's also usual that the application managers want to see what have changed in schema, so we'll give them also the SELECT permission

-- create a custom role "ApplicationEndUsers"

insert into security.DatabaseRoles (

ServerName,

DbName,

RoleName,

isStandard,

isActive,

Reason

)

values (

'MyServer1',

'ApplicationDatabase1',

'ApplicationEndUsers',

0,

1,

'Group all \*\_endusers in one role'

)

-- assign \*\_endusers role membership to "ApplicationEndusers"

insert into security.DatabaseRoleMembers (

ServerName,

DbName,

RoleName,

MemberName,

MemberIsRole,

PermissionLevel,

Reason,

isActive

)

values

('MyServer1','ApplicationDatabase1','ApplicationSchema1\_endusers','ApplicationEndusers',1,'GRANT','Group all \*\_endusers in one role',1 ),

('MyServer1','ApplicationDatabase1','ApplicationSchema2\_endusers','ApplicationEndusers',1,'GRANT','Group all \*\_endusers in one role',1 )

-- create a custom role "ApplicationResponsibles"

insert into security.DatabaseRoles (

ServerName,

DbName,

RoleName,

isStandard,

isActive,

Reason

)

values (

'MyServer1',

'ApplicationDatabase1',

'ApplicationResponsibles',

0,

1,

'Group all \*\_responsible in one role'

)

-- assign \*\_responsible role membership to "ApplicationResponsibles"

insert into security.DatabaseRoleMembers (

ServerName,

DbName,

RoleName,

MemberName,

MemberIsRole,

PermissionLevel,

Reason,

isActive

)

values

('MyServer1','ApplicationDatabase1','ApplicationSchema1\_responsible','ApplicationResponsibles',1,'GRANT','Group all \*\_responsible in one role',1 ),

('MyServer1','ApplicationDatabase1','ApplicationSchema2\_responsible','ApplicationResponsibles',1,'GRANT','Group all \*\_responsible in one role',1 )

-- 4. INSERT + SELECT on 'dbo.SchemaChangeLog' to ApplicationResponsibles role

insert into security.DatabasePermissions (

ServerName,

DbName,

Grantee,

isUser,

ObjectClass,

ObjectType,

PermissionLevel,

PermissionName,

SchemaName,

ObjectName,

Reason,

isActive

)

values

('MyServer1','ApplicationDatabase1','ApplicationResponsibles',0,'SCHEMA\_OBJECT','TABLE','GRANT','SELECT','dbo','SchemaChangeLog','Allow responsibles to query DDL changes history',1),

('MyServer1','ApplicationDatabase1','ApplicationResponsibles',0,'SCHEMA\_OBJECT','TABLE','GRANT','INSERT','dbo','SchemaChangeLog','Allow responsibles to execute DDL changes (otherwize, they are not allowed to do it)',1)

**Note** : in my case, the point 4 should be done also for each \*\_struct\_modifier. I'm planning on creating a helper procedure which will do those things for me.

**Script Generation**

Now, we have a full set of data ready for generation. Let's check out how to do it. This part will maybe change a little bit in the future as it takes as key a datetime...

-- Historize generation + display as table

exec [security].[getSecurityScript]

@ServerName = 'MyServer1',

@DbName = NULL,

@AsOf = NULL ,

@OutputType = 'TABLE' ,

@OutputDatabaseName = 'DbaDoc' ,

@OutputSchemaName = 'dbo' ,

@OutputTableName = 'GeneratedSecurityScripts' ,

@Debug = 1

Here is a way to get directly a script to execute AND historize generation

exec [security].[getSecurityScript]

@ServerName = 'MyServer1',

@DbName = NULL,

@AsOf = NULL ,

@OutputType = 'SCRIPT' ,

@OutputDatabaseName = 'DbaDoc' ,

@OutputSchemaName = 'dbo' ,

@OutputTableName = 'GeneratedSecurityScripts' ,

@Debug = 1

A sample generated script is attached to this article.

**Generated script code review**

Let's have a look at the generated code. It's attached to this article.

The first part of the script is a set of comments. It's the log of the generator. As we set @Debug to 1, there are DEBUG comments.

-- Jan 11 2015 6:35PM-- DEBUG - Table ##SecurityGenerationResults dropped

-- Jan 11 2015 6:35PM-- DEBUG - Table ##SecurityScriptResultsCommandsOrder dropped

-- Jan 11 2015 6:35PM-- DEBUG - Table ##SecurityGenerationResults created

-- Jan 11 2015 6:35PM-- DEBUG - Table ##SecurityScriptResultsCommandsOrder created

-- Jan 11 2015 6:35PM - DEBUG - > Login creation statements

-- Jan 11 2015 6:35PM - DEBUG - Every logins generation detected.

-- Jan 11 2015 6:35PM - DEBUG - Taking care of login Enduser1 (Application End User 1)

-- Jan 11 2015 6:35PM - DEBUG - Taking care of login MyDomain\Jeff.Elias (Jefferson Elias)

-- Jan 11 2015 6:35PM - DEBUG - Taking care of login MyDomain\JohnDoe (John Doe)

-- Jan 11 2015 6:35PM - DEBUG - Full server generation mode detected.

--------------------------------------------------------------------------------------------------------------

-- Jan 11 2015 6:35PM - DEBUG - Server Name : MyServer1

-- Jan 11 2015 6:35PM - DEBUG - Database set to ApplicationDatabase1

-- Jan 11 2015 6:35PM - DEBUG - > Schema creation statements

-- Jan 11 2015 6:35PM - DEBUG - Every Schema generation detected.

-- Jan 11 2015 6:35PM - DEBUG - > Database user creation statements

-- Jan 11 2015 6:35PM - DEBUG - Every users in database generation detected.

-- Jan 11 2015 6:35PM - DEBUG - ServerName MyServer1

-- Jan 11 2015 6:35PM - DEBUG - DbName ApplicationDatabase1

-- Jan 11 2015 6:35PM - DEBUG - Taking care of user Enduser1

-- Jan 11 2015 6:35PM - DEBUG - ServerName MyServer1

-- Jan 11 2015 6:35PM - DEBUG - DbName ApplicationDatabase1

-- Jan 11 2015 6:35PM - DEBUG - Taking care of user MyDomain\JohnDoe

-- Jan 11 2015 6:35PM - DEBUG - > Login to database user creation statements

-- Jan 11 2015 6:35PM - DEBUG - All mappings for database ApplicationDatabase1 detected

-- Jan 11 2015 6:35PM - DEBUG - > Database Roles creation and assignment statements

-- Jan 11 2015 6:35PM - DEBUG - Every Role generation detected.

-- Jan 11 2015 6:35PM - DEBUG - Every Role membership generation detected.

-- Jan 11 2015 6:35PM - DEBUG - > Object-level permission assignment statements

-- Jan 11 2015 6:35PM - DEBUG - Every permission assignment detected.

-- Jan 11 2015 6:35PM - DEBUG - Taking care of permissions for [ApplicationResponsibles] in database [ApplicationDatabase1]

-- Jan 11 2015 6:35PM - DEBUG - Taking care of permissions for [ApplicationSchema1\_data\_modifier] in database [ApplicationDatabase1]

-- Jan 11 2015 6:35PM - DEBUG - Taking care of permissions for [ApplicationSchema1\_data\_reader] in database [ApplicationDatabase1]

-- Jan 11 2015 6:35PM - DEBUG - Taking care of permissions for [ApplicationSchema1\_prog\_executors] in database [ApplicationDatabase1]

-- Jan 11 2015 6:35PM - DEBUG - Taking care of permissions for [ApplicationSchema1\_struct\_modifier] in database [ApplicationDatabase1]

-- Jan 11 2015 6:35PM - DEBUG - Taking care of permissions for [ApplicationSchema1\_struct\_viewer] in database [ApplicationDatabase1]

-- Jan 11 2015 6:35PM - DEBUG - Taking care of permissions for [ApplicationSchema2\_data\_modifier] in database [ApplicationDatabase1]

-- Jan 11 2015 6:35PM - DEBUG - Taking care of permissions for [ApplicationSchema2\_data\_reader] in database [ApplicationDatabase1]

-- Jan 11 2015 6:35PM - DEBUG - Taking care of permissions for [ApplicationSchema2\_prog\_executors] in database [ApplicationDatabase1]

-- Jan 11 2015 6:35PM - DEBUG - Taking care of permissions for [ApplicationSchema2\_struct\_modifier] in database [ApplicationDatabase1]

-- Jan 11 2015 6:35PM - DEBUG - Taking care of permissions for [ApplicationSchema2\_struct\_viewer] in database [ApplicationDatabase1]

-- Jan 11 2015 6:35PM - INFO - Generation successful

--------------------------------------------------------------------------------------------------------------

-- Nothing to display

--------------------------------------------------------------------------------------------------------------

-- Jan 11 2015 6:35PM - DEBUG - Server Name : MyServer1

-- Jan 11 2015 6:35PM - DEBUG - Database set to master

-- Jan 11 2015 6:35PM - DEBUG - > Schema creation statements

-- Jan 11 2015 6:35PM - DEBUG - Every Schema generation detected.

-- Jan 11 2015 6:35PM - DEBUG - > Database user creation statements

-- Jan 11 2015 6:35PM - DEBUG - Every users in database generation detected.

-- Jan 11 2015 6:35PM - DEBUG - ServerName MyServer1

-- Jan 11 2015 6:35PM - DEBUG - DbName master

-- Jan 11 2015 6:35PM - DEBUG - Taking care of user MyDomain\Jeff.Elias

-- Jan 11 2015 6:35PM - DEBUG - > Login to database user creation statements

-- Jan 11 2015 6:35PM - DEBUG - All mappings for database master detected

-- Jan 11 2015 6:35PM - DEBUG - > Database Roles creation and assignment statements

-- Jan 11 2015 6:35PM - DEBUG - Every Role generation detected.

-- Jan 11 2015 6:35PM - DEBUG - Every Role membership generation detected.

-- Jan 11 2015 6:35PM - DEBUG - > Object-level permission assignment statements

-- Jan 11 2015 6:35PM - DEBUG - Every permission assignment detected.

-- Jan 11 2015 6:35PM - INFO - Generation successful

--------------------------------------------------------------------------------------------------------------

-- Nothing to display

-- Jan 11 2015 6:35PM - DEBUG - Creating table (if not exists) where we must save results

-- Jan 11 2015 6:35PM - DEBUG - Filling in with results

--------------------------------------------------------------------------------------------------------------

Then the generated script starts. First, by a set of checks

IF (@@SERVERNAME <> 'MyServer1')

BEGIN

RAISERROR('Expected @@ServerName : "MyServer1"', 16, 0)

END

IF NOT EXISTS( SELECT 1 FROM sys.databases where Name = 'ApplicationDatabase1')

BEGIN

RAISERROR('The Database named : "ApplicationDatabase1" doesn''t exist on server !', 16, 0)

END

IF NOT EXISTS( SELECT 1 FROM sys.databases where Name = 'master')

BEGIN

RAISERROR('The Database named : "master" doesn''t exist on server !', 16, 0)

END

Then the effective T-SQL commands to execute against the server MyServer1.

PRINT '. Commands for "Application End User 1" from department "External/DevCorp/DevProduct"'

DECLARE @loginToPlayWith VARCHAR(64)

SET @loginToPlayWith = QUOTENAME('Enduser1')

IF NOT EXISTS (SELECT name FROM master.sys.server\_principals WHERE QUOTENAME(name) = @loginToPlayWith)

BEGIN

-- create it !

EXEC ('USE [master]; CREATE LOGIN [Enduser1] WITH PASSWORD=N''123456a.'' , DEFAULT\_DATABASE=[ApplicationDatabase1]')

END

GO

etc.

**What's next ?**

This solution is provided for new SQL Server installations. But, many of us also have existing installations. The next part that will be to:

1. Give the ability to manage an existing documented server which has already been implemented.
2. For removal, I think about a workaround : a) set revoke permission, b) generate script c) execute script d) delete any reference to what we removed
3. Build an extraction tool of the actual security on a server and a procedure to merge both so that we get something cool.
4. Once it's done, I'd like to add the database and server level object permission assignment in order to fullfill the solution.
5. In the end, I'd like to add a GUI (even if I'm not a developper) to this tool so that this will be very easy to use.

If you want to collaborate on this project, don't hesitate to contact me. I'll be thankful for that.

**Acknowledgments**

I want to thank Brent Ozar for his script sp\_AskBrent, which inspired me for the way I generate the code to execute on the server.

I also want to thank Nicholas C. Zakas, the author of a tool named "[Combiner](http://www.nczonline.net/blog/2009/09/22/introducing-combiner-a-javascriptcss-concatenation-tool/)" which I use to generate a single script for distribution, as the one attached to this article.

Attached script is in C:\WJB\_SQL\_Scripts\ securityApplier\_SQLServer\_1.0.1.sql