**Apply Row-Level Security to all tables — helper script**

★★★★★

★★★★

★★★

★★

★

March 30, 2015 by [Tommy Mullaney](https://social.msdn.microsoft.com/profile/Tommy+Mullaney)

Tommy Mullaney

Microsoft

}

MSFT

Developing multi-tenant applications with Row-Level Security (RLS) just got a little easier. This post makes available a script that will automatically apply an RLS predicate to all tables in a database.

Applications with multi-tenant databases, including those using [Elastic Scale](http://azure.microsoft.com/en-us/documentation/articles/sql-database-elastic-scale-introduction/) for sharding, commonly have a “TenantId” column in every table to indicate which rows belong to each tenant. As described in [Building More Secure Middle-Tier Applications with Azure SQL Database using Row-Level Security](http://azure.microsoft.com/blog/2015/03/02/building-more-secure-middle-tier-applications-with-azure-sql-database-using-row-level-security/), the recommended approach for filtering out rows that don’t belong to a tenant querying the database is to create an RLS security policy that filters out rows whose TenantId doesn’t match the current value of CONTEXT\_INFO. However, for large applications with perhaps hundreds of tables, it can be tedious to write out “ADD FILTER PREDICATE…” for every table when creating or altering the RLS security policy.

To streamline this common RLS use case, we’ve created a helper stored procedure to automatically generate a security policy that adds a filter predicate on all tables with a TenantId column. See below for syntax and three common usage examples.

Script available here: <http://rlssamples.codeplex.com/SourceControl/latest#RLS-Auto-Enable.sql>

**Syntax:**

CREATE PROCEDURE dbo.sp\_enable\_rls\_auto (  
 /\* The type for the tenant ID column. It could be short, int or bigint. \*/  
 @rlsColType sysname,   
   
 /\* The name for the tenant ID column. All tables that match the column name & type will be affected. \*/  
 @rlsColName sysname,  
   
 /\* The schema name where the policy will be applied.  
 If null (default), the policy will be applied to tables in all schemas in the database. \*/  
 @applyToSchema sysname = null,  
   
 /\* Set to 1 to disable all existing policies that affect the identified target tables.  
 If set to 0 (default), this function will fail if there is an existing policy on any of these tables. \*/  
 @deactivateExistingPolicies bit = 0,  
   
 /\* Schema name for new RLS objects. If it does not exist, it will be created. \*/  
 @rlsSchemaName sysname = N'rls',  
   
 /\* The name of an existing function in the RLS schema that will be used as the predicate.  
 If null (default), a new function will be created with a simple CONTEXT\_INFO = tenant ID filter. \*/  
 @rlsPredicateFunctionName sysname = null,  
   
 /\* Set to 1 to allow CONTEXT\_INFO = null to have access to all rows. Default is 0.  
 Not applicable if @rlsPredicateFunctionName is set with a custom predicate function. \*/  
 @isNullAdmin bit = 0,  
   
 /\* If @isNullAdmin = 1, set to 1 to optimize the CONTEXT\_INFO = null disjunction into a range query.   
 Not applicable if @rlsPredicateFunctionName is set with a custom predicate function.   
 Note that on Azure SQL Database, CONTEXT\_INFO is pre-populated with a unique connection GUID (not null),   
 so you must execute SET CONTEXT\_INFO 0x to reset it to null for this 'admin' mode to work. \*/  
 @isNullAdminOptimized bit = 1,  
   
 /\* If set, the predicate function will allow only this user to access rows.  
 Use only for middle-tier scenarios, where this is the shared application user name.   
 Not applicable if @rlsPredicateFunctionName is set with a custom predicate function. \*/  
 @restrictedToAppUserName sysname = null,  
   
 /\* Set to 1 to print the commands (on by default). \*/  
 @printCommands bit = 1,  
   
 /\* Set to 1 to execute the commands (off by default). \*/  
 @runCommands bit = 0  
)

**Examples:**

**Example 1: Typical CONTEXT\_INFO usage**

Generate a security policy that adds a new filter predicate (using CONTEXT\_INFO as described in [Building More Secure Middle-Tier Applications with Azure SQL Database using Row-Level Security](http://azure.microsoft.com/blog/2015/03/02/building-more-secure-middle-tier-applications-with-azure-sql-database-using-row-level-security/)) on all tables with a “TenantId” column of type “int.” Only allow access to “AppUser,” the shared application user in our app’s connection string. If CONTEXT\_INFO is null, filter all rows by default.

EXEC sp\_enable\_rls\_auto  
 @rlsColType = 'int',   
 @rlsColName = 'TenantId',  
 @applyToSchema = null,  
 @deactivateExistingPolicies = 1,  
 @rlsSchemaName = N'rls',  
 @rlsPredicateFunctionName = null,  
 @isNullAdmin = 0,  
 @isNullAdminOptimized = 0,  
 @restrictedToAppUserName = 'AppUser',  
 @printCommands = 1,  
 @runCommands = 0 -- set to 1 to execute output  
go

Sample output:

CREATE FUNCTION [rls].[fn\_predicate\_TenantId\_2015-03-30T17:36:51.010](@TenantId [int] )  
 RETURNS TABLE  
 WITH SCHEMABINDING  
AS  
 RETURN SELECT 1 AS fn\_accessResult   
 WHERE   
 DATABASE\_PRINCIPAL\_ID() = DATABASE\_PRINCIPAL\_ID ('AppUser')   
 AND CONVERT([int], CONVERT(varbinary(4), CONTEXT\_INFO())) = @TenantId  
go  
  
CREATE SECURITY POLICY [rls].[secpol\_TenantId\_2015-03-30T17:36:51.073]  
 ADD FILTER PREDICATE [rls].[fn\_predicate\_TenantId\_2015-03-30T17:36:51.010]([TenantId]) ON [dbo].[Sales],  
 ADD FILTER PREDICATE [rls].[fn\_predicate\_TenantId\_2015-03-30T17:36:51.010]([TenantId]) ON [dbo].[Products],  
 ADD FILTER PREDICATE [rls].[fn\_predicate\_TenantId\_2015-03-30T17:36:51.010]([TenantId]) ON [dbo].[PriceHistory],  
 ADD FILTER PREDICATE [rls].[fn\_predicate\_TenantId\_2015-03-30T17:36:51.010]([TenantId]) ON [dbo].[OrderDetails]  
go

**Example 2: Custom predicate function**

Generate a security policy that adds a custom predicate function as a filter predicate on all tables with a “TenantId” column of type “int.”

CREATE FUNCTION rls.customTenantAccessPredicate(@TenantId int)  
 RETURNS TABLE  
 WITH SCHEMABINDING  
AS  
RETURN SELECT 1 AS accessResult WHERE  
(  
 DATABASE\_PRINCIPAL\_ID() = DATABASE\_PRINCIPAL\_ID('AppUser') -- shared app user  
 AND CONVERT(int, CONVERT(varbinary(4), CONTEXT\_INFO())) = @TenantId  
)   
OR  
 DATABASE\_PRINCIPAL\_ID() = DATABASE\_PRINCIPAL\_ID('ReportUser') -- reporting user can see all rows  
go  
   
EXEC sp\_enable\_rls\_auto  
 @rlsColType = 'int',   
 @rlsColName = 'TenantId',  
 @applyToSchema = null,  
 @deactivateExistingPolicies = 1,  
 @rlsSchemaName = N'rls',  
 @rlsPredicateFunctionName = N'customTenantAccessPredicate',  
 @isNullAdmin = 0, -- n/a  
 @isNullAdminOptimized = 0, -- n/a  
 @restrictedToAppUserName = null, -- n/a  
 @printCommands = 1,  
 @runCommands = 0 -- set to 1 to execute output  
go

Sample output:

CREATE SECURITY POLICY [rls].[secpol\_TenantId\_2015-03-30T18:22:14.213]  
 ADD FILTER PREDICATE [rls].[customTenantAccessPredicate]([TenantId]) ON [dbo].[Sales],  
 ADD FILTER PREDICATE [rls].[customTenantAccessPredicate]([TenantId]) ON [dbo].[Products],  
 ADD FILTER PREDICATE [rls].[customTenantAccessPredicate]([TenantId]) ON [dbo].[PriceHistory],  
 ADD FILTER PREDICATE [rls].[customTenantAccessPredicate]([TenantId]) ON [dbo].[OrderDetails]   
go

**Example 3: Optimized “superuser” if CONTEXT\_INFO is null**

Same as Example 1, but if CONTEXT\_INFO is null, make all rows visible to the application and utilize the performance optimization for disjunctions described in [Row-Level Security for Middle-Tier Apps – Using Disjunctions in the Predicate](http://blogs.msdn.com/b/sqlsecurity/archive/2015/03/16/row-level-security-for-middle-tier-apps-using-disjunctions-in-the-predicate.aspx).

EXEC sp\_enable\_rls\_auto  
 @rlsColType = 'int',   
 @rlsColName = 'TenantId',  
 @applyToSchema = null,  
 @deactivateExistingPolicies = 1,  
 @rlsSchemaName = N'rls',  
 @rlsPredicateFunctionName = null,  
 @isNullAdmin = 1,  
 @isNullAdminOptimized = 1,  
 @restrictedToAppUserName = 'AppUser',  
 @printCommands = 1,  
 @runCommands = 0 -- set to 1 to execute output  
go

Sample output:

CREATE FUNCTION [rls].[int\_lo\_2015-03-30T18:30:46.993]() RETURNS [int]  
 WITH SCHEMABINDING   
AS BEGIN   
 RETURN CASE WHEN context\_info() is null THEN  
 -2147483648 ELSE   
 convert([int], convert(varbinary(4), context\_info())) END   
END  
go  
   
CREATE FUNCTION [rls].[int\_hi\_2015-03-30T18:30:46.993]() RETURNS [int]  
 WITH SCHEMABINDING   
AS BEGIN   
 RETURN CASE WHEN context\_info() is null THEN  
 2147483647 ELSE   
 convert([int], convert(varbinary(4), context\_info())) END   
END  
go  
   
CREATE FUNCTION [rls].[fn\_predicate\_TenantId\_2015-03-30T18:30:46.993](@TenantId [int] )  
 RETURNS TABLE  
 WITH SCHEMABINDING  
AS  
RETURN SELECT 1 AS fn\_accessResult   
WHERE   
 DATABASE\_PRINCIPAL\_ID() = DATABASE\_PRINCIPAL\_ID ('AppUser') AND (  
 (@TenantId BETWEEN [rls].[int\_lo\_2015-03-30T18:30:46.993]() AND [rls].[int\_hi\_2015-03-30T18:30:46.993]())  
go  
   
CREATE SECURITY POLICY [rls].[secpol\_TenantId\_2015-03-30T18:30:47.047]  
 ADD FILTER PREDICATE [rls].[fn\_predicate\_TenantId\_2015-03-30T18:30:46.993]([TenantId]) ON [dbo].[Sales],  
 ADD FILTER PREDICATE [rls].[fn\_predicate\_TenantId\_2015-03-30T18:30:46.993]([TenantId]) ON [dbo].[Products],  
 ADD FILTER PREDICATE [rls].[fn\_predicate\_TenantId\_2015-03-30T18:30:46.993]([TenantId]) ON [dbo].[PriceHistory],  
 ADD FILTER PREDICATE [rls].[fn\_predicate\_TenantId\_2015-03-30T18:30:46.993]([TenantId]) ON [dbo].[OrderDetails]  
go

**Recent Posts**

* [PowerShell cmdlets for managing SQL Vulnerability Assessments](https://blogs.msdn.microsoft.com/sqlsecurity/2018/07/05/powershell-cmdlets-for-managing-sql-vulnerability-assessments/) July 5, 2018
* [Introducing SQL Information Protection for Azure SQL Database and on-premises SQL Server!](https://blogs.msdn.microsoft.com/sqlsecurity/2018/02/20/introducing-sql-information-protection-for-azure-sql-database-and-on-premises-sql-server/) February 20, 2018
* [Azure Log Analytics (OMS) Agent now collects SQL Server audit logs](https://blogs.msdn.microsoft.com/sqlsecurity/2017/12/28/azure-log-analytics-oms-agent-now-collects-sql-server-audit-logs/) December 28, 2017
* [SQL Vulnerability Assessment now available for SQL Server 2012 and up](https://blogs.msdn.microsoft.com/sqlsecurity/2017/12/11/sql-vulnerability-assessment-now-available-for-sql-server-2012-and-up/) December 11, 2017

https://secure.gravatar.com/avatar/3651e719491c63d4424134d66fe53db9?s=40&d=mm&r=g

Francesc

This looks great, but shouldn't it use the new SESSION\_CONTEXT instead of CONTEXT\_INFO?

* + 2 years ago

https://secure.gravatar.com/avatar/ea39bbcb64ea97add9154fc192013e69?s=40&d=mm&r=g

Mark

Bump… I second this question.