EXECUTE AS Clause (Transact-SQL)

Source	http://msdn2.microsoft.com/en-us/library/ms188354.aspx

EXECUTE AS Clause (Transact-SQL)

In SQL Server 2005 you can define the execution context of the following user-defined modules: functions (except inline table-valued functions), procedures, queues, and triggers.

By specifying the context in which the module is executed, you can control which user account the SQL Server 2005 Database Engine uses to validate permissions on objects that are referenced by the module. This provides additional flexibility and control in managing permissions across the object chain that exists between user-defined modules and the objects referenced by those modules. Permissions must be granted to users only on the module itself, without having to grant them explicit permissions on the referenced objects. Only the user that the module is running as must have permissions on the objects accessed by the module.

Syntax

```
Functions (except inline table-valued functions), Stored Procedures,
and DML Triggers

{ EXEC | EXECUTE } AS { CALLER | SELF | OWNER | 'user_name' }

DDL Triggers with Database Scope

{ EXEC | EXECUTE } AS { CALLER | SELF | 'user_name' }

DDL Triggers with Server Scope and logon triggers

{ EXEC | EXECUTE } AS { CALLER | SELF | 'login_name' }

Queues

{ EXEC | EXECUTE } AS { SELF | OWNER | 'user_name' }
```

Arguments

CALLER

Specifies the statements inside the module are executed in the context of the caller of the module. The user executing the module must have appropriate permissions not only on the module itself, but also on any database objects that are referenced by the module.

CALLER is the default for all modules except queues, and is the same as SQL Server 2000 behavior.

CALLER cannot be specified in a CREATE QUEUE or ALTER QUEUE statement.

SELF

EXECUTE AS SELF is equivalent to EXECUTE AS *user_name*, where the specified user is the person creating or altering the module. The actual user ID of the person creating or modifying the modules is stored in the execute_as_principal_id column in the sys.sql_modules or sys.service_queues catalog view.

SELF is the default for queues.

Note: To change the user ID of the execute_as_principal_id in the sys.service_queues catalog view, you must explicitly specify the EXECUTE AS setting in the ALTER QUEUE statement.

OWNER

Specifies the statements inside the module executes in the context of the current owner of the module. If the module does not have a specified owner, the owner of the schema of the module is used. OWNER cannot be specified for DDL triggers.

Important: OWNER must map to a singleton account and cannot be a role or group.

' user_name '

Specifies the statements inside the module execute in the context of the user specified in user_name. Permissions for any objects within the module are verified against user_name. user_name cannot be specified for DDL triggers with server scope. Use login_name instead.

user_name must exist in the current database and must be a singleton account. user_name cannot be a group, role, certificate, key, or built-in account, such as NT AUTHORITY\LocalService, NT AUTHORITY\NetworkService, or NT AUTHORITY\LocalSystem.

The user ID of the execution context is stored in metadata and can be viewed in the execute_as_principal_id column in the sys.sql_modules or sys.assembly_modules catalog view.

' login_name '

Specifies the statements inside the module execute in the context of the SQL Server login specified in login_name. Permissions for any objects within the module are verified against login_name. login_name can be specified only for DDL triggers with server scope.

login_name cannot be a group, role, certificate, key, or built-in account, such as NT AUTHORITY\LocalService, NT AUTHORITY\NetworkService, or NT AUTHORITY\LocalSystem.

Remarks

How the Database Engine evaluates permissions on the objects that are referenced in the module depends on the ownership chain that exists between calling objects and referenced objects. In earlier versions of SQL Server, ownership chaining was the only method available to avoid having to grant the calling user access to all referenced objects.

Ownership chaining has the following limitations:

- **Applies only to DML statements: SELECT, INSERT, UPDATE, and DELETE.**
- **The owners of the calling and the called objects must be the same.**
- **Does not apply to dynamic queries inside the module.**

For more information about ownership chaining, see Ownership Chains.

Regardless of the execution context that is specified in the module, the following actions always apply:

- **When the module is executed, the Database Engine first verifies that the user executing the module has EXECUTE permission on the module.**
- **Ownership chaining rules continue to apply. This means if the owners of the calling and called objects are the same, no permissions are checked on the underlying objects.**

When a user executes a module that has been specified to run in a context other than CALLER, the user's permission to execute the module is checked, but additional permissions checks on objects that are accessed by the module are performed against the user account specified in the EXECUTE AS clause. The user executing the module is, in effect, impersonating the specified user.

The context specified in the EXECUTE AS clause of the module is valid only for the duration of the module execution. Context reverts to the caller when the module execution is completed. For more information about switching execution context in a module, see Using EXECUTE AS in Modules.

Specifying a User or Login Name

A database user or server login specified in the EXECUTE AS clause of a module cannot be dropped until the module has been modified to execute under another context.

The user or login name specified in EXECUTE AS clause must exist as a principal in sys.database_principals or sys.server_principals, respectively, or else the create or alter module operation fails. Additionally, the user that creates or alters the module must have IMPERSONATE permissions on the principal.

If the user has implicit access to the database or instance of SQL Server through a Windows group membership, the user specified in the EXECUTE AS clause is implicitly created when the module is created when one of the following requirements exist:

- **The specified user or login is a member of the sysadmin fixed server role.**
- **The user that is creating the module has permission to create principals.**

When neither of these requirements are met, the create module operation fails.

Important:

If the SQL Server (MSSQLSERVER) service is running as a local account (local service or local user account), it will not have privileges to obtain the group memberships of a Windows domain account that is specified in the EXECUTE AS clause. This will cause the execution of the module to fail.

For example, assume the following conditions:

- **CompanyDomain\SQLUsers group has access to the Sales database.**
- **CompanyDomain\SqlUser1 is a member of SQLUsers and, therefore, has access to the Sales database.**
- **The user that is creating or altering the module has permissions to create principals.**

When the following CREATE PROCEDURE statement is run, the CompanyDomain\SqlUser1 is implicitly created as a database principal in the Sales database.

```
USE Sales;

GO

CREATE PROCEDURE dbo.usp_Demo

WITH EXECUTE AS 'CompanyDomain\SqlUser1'

AS

SELECT user_name();

GO
```

Using EXECUTE AS CALLER Stand-alone Statement

Use the EXECUTE AS CALLER stand-alone statement inside a module to set the execution context to the caller of the module.

Assume the following stored procedure is called by SqlUser2.

```
CREATE PROCEDURE dbo.usp_Demo
```

```
WITH EXECUTE AS 'SqlUser1'

AS

SELECT user_name(); -- Shows execution context is set to SqlUser1.

EXECUTE AS CALLER;

SELECT user_name(); -- Shows execution context is set to SqlUser2, the
caller of the module.

REVERT;

SELECT user_name(); -- Shows execution context is set to SqlUser1.

GO
```

Using EXECUTE AS to Define Custom Permission Sets

Specifying an execution context for a module can be very useful when you want to define custom permission sets. For example, some actions, such as TRUNCATE TABLE, do not have grantable permissions. By incorporating the TRUNCATE TABLE statement within a module and specifying that module execute as a user who has permissions to alter the table, you can extend the permissions to truncate the table to the user to whom you grant EXECUTE permissions on the module. For more information, see Using EXECUTE AS to Create Custom Permission Sets.

To view the definition of the module with the specified execution context, use the sys.sql_modules (Transact-SQL) catalog view.

Best Practice

Specify a login or user that has the least privileges required to perform the operations defined in the module. For example, do not specify a database owner account unless those permissions are required.

Permissions

To execute a module specified with EXECUTE AS, the caller must have EXECUTE permissions on the module.

To execute a CLR module specified with EXECUTE AS that accesses resources in another database or server, the target database or server must trust the authenticator of the database from which the module originates (the source database). For more information about how to establish authenticator trust, see Extending Database Impersonation by Using EXECUTE AS.

To specify the EXECUTE AS clause when you create or modify a module, you must have IMPERSONATE permissions on the specified principal and also permissions to create the module. You can always impersonate yourself. When no execution context is specified or EXECUTE AS CALLER is specified, IMPERSONATE permissions are not required.

To specify a *login_name* or *user_name* that has implicit access to the database through a Windows group membership, you must have CONTROL permissions on the database.

Examples

The following example creates a stored procedure and assigns the execution context to OWNER.

```
USE AdventureWorks;

GO

CREATE PROCEDURE HumanResources.uspEmployeesInDepartment

@DeptValue int

WITH EXECUTE AS OWNER

AS

SELECT e.EmployeeID, c.LastName, c.FirstName, e.Title

FROM Person.Contact AS c

INNER JOIN HumanResources.Employee AS e

    ON c.ContactID = e.ContactID

INNER JOIN HumanResources.EmployeeDepartmentHistory AS edh
```

```
    ON e.EmployeeID = edh.EmployeeID
WHERE edh.DepartmentID = @DeptValue
ORDER BY c.LastName, c.FirstName;
GO
-- Execute the stored procedure by specifying department 5.
EXECUTE HumanResources.uspEmployeesInDepartment 5;
GO
```

~~~ End of Article ~~~