**How to Setup Row Level Security for SQL Server**

**Problem**

I've been tasked to put together a row-level security model for a database where users are connecting with their Windows user accounts. I'm not sure how to go about this. How do I setup such a security model?

**Solution**

This is actually not as hard as it sounds, especially if your organization is making proper use of Windows groups in your Active Directory domain(s). For instance, let's consider a sample database that has employee information. Here's how we'd break down security:

* The DBAs obviously have access to all the rows.
* HR personnel also need access to all the rows.
* Managers only have access to employee information for their departments.

This type of scenario is perfect for instituting a row-level security solution. So let's walk through how to do this.

First, let's set up some basic tables and load them with data.

USE MSSQLTips;
GO

-- Sample tables to use to demonstrate row-level security
-- We'll key off the DepartmentID as our filter
CREATE TABLE dbo.Department (
DepartmentID INT NOT NULL,
DepartmentName VARCHAR(30),
CONSTRAINT PK\_Department PRIMARY KEY CLUSTERED (DepartmentID)
);
GO

CREATE TABLE dbo.Employee (
EmployeeID INT NOT NULL,
FirstName VARCHAR(30) NOT NULL,
MiddleName VARCHAR(30) NULL,
SurName VARCHAR(30) NOT NULL,
DepartmentID INT NOT NULL,
CONSTRAINT PK\_Employee PRIMARY KEY CLUSTERED (EmployeeID),
CONSTRAINT FK\_Employee\_Department FOREIGN KEY (DepartmentID) REFERENCES dbo.Department (DepartmentID)
);
GO

-- Sample data so we can verify the row-level security will work as we modeled it
INSERT INTO dbo.Department (DepartmentID, DepartmentName) VALUES (1, 'IT');
INSERT INTO dbo.Department (DepartmentID, DepartmentName) VALUES (2, 'HR');
INSERT INTO dbo.Department (DepartmentID, DepartmentName) VALUES (3, 'Marketing');
INSERT INTO dbo.Department (DepartmentID, DepartmentName) VALUES (4, 'Facilities');
INSERT INTO dbo.Department (DepartmentID, DepartmentName) VALUES (5, 'Accounting');
GO

INSERT INTO dbo.Employee (EmployeeID, FirstName, MiddleName, SurName, DepartmentID) VALUES (1, 'George', NULL, 'Washington', 1);
INSERT INTO dbo.Employee (EmployeeID, FirstName, MiddleName, SurName, DepartmentID) VALUES (2, 'John', NULL, 'Adams', 2);
INSERT INTO dbo.Employee (EmployeeID, FirstName, MiddleName, SurName, DepartmentID) VALUES (3, 'Thomas', NULL, 'Jefferson', 3);
INSERT INTO dbo.Employee (EmployeeID, FirstName, MiddleName, SurName, DepartmentID) VALUES (4, 'James', NULL, 'Madison', 4);
INSERT INTO dbo.Employee (EmployeeID, FirstName, MiddleName, SurName, DepartmentID) VALUES (5, 'James', NULL, 'Monroe', 5);
INSERT INTO dbo.Employee (EmployeeID, FirstName, MiddleName, SurName, DepartmentID) VALUES (6, 'John', 'Quincy', 'Adams', 1);
INSERT INTO dbo.Employee (EmployeeID, FirstName, MiddleName, SurName, DepartmentID) VALUES (7, 'Andrew', NULL, 'Jackson', 2);
INSERT INTO dbo.Employee (EmployeeID, FirstName, MiddleName, SurName, DepartmentID) VALUES (8, 'Martin', 'Van', 'Buren', 3);
INSERT INTO dbo.Employee (EmployeeID, FirstName, MiddleName, SurName, DepartmentID) VALUES (9, 'William', 'Henry', 'Harrison', 4);
INSERT INTO dbo.Employee (EmployeeID, FirstName, MiddleName, SurName, DepartmentID) VALUES (10, 'John', NULL, 'Tyler', 5);
GO

So we have 5 departments, with 2 employees per department. We'd expect a DBA or an HR person to see all 10 employees. We'd only expect a manager to see 2 employees. However, if we just give SELECT access to the tables, then a manager could query the whole table and get all the rows back. This isn't what we want.

We've got to build a mechanism on top of these tables. That mechanism has to somehow tie to the Windows groups to filter the data properly. So what we're going to need is some sort of security mapping table that ties our security to the department as well as a view that uses that security mapping table to properly filter the results. We'll use ownership chaining, granting SELECT permissions against the view, but not granting any permissions against the underlying tables. This will force the managers to come through the view, and if our view is written right, row-level security will be enforced.

The question then is, what should the security mapping table look like? We know we need DepartmentID, because we're filtering on department. We could use the Windows group name, but I prefer to create database roles that mirror what we'd expect to find. There's a couple of reasons for this. First, if I just want to test my model and I don't have the Windows groups yet, I can tie things to the database roles and add and remove users from the database roles to verify my security model is working. Second, if you switch domains between production and development, you won't have to worry about re-seeding your security mapping table, you'll just need to associate the right Windows security groups to the right database roles. You'll have to touch these Windows groups to make them users anyway, so this is probably the solution that requires the least amount of work.

For our example, we can create a database role for HR personnel, and one for each manager type. Since we're not doing a full blown implementation, let's create 1 for the HR personnel, which we'll call HR\_User, and create several roles for different managers. Three roles should be sufficient to show us that row-level security is working. We also want to create a generic role, Manager, which is what we'll use to give permissions to select against the view, once we create it.

-- Create database roles to go along with the Windows groups
CREATE ROLE HR\_User;
GO

CREATE ROLE IT\_Manager;
GO

CREATE ROLE Marketing\_Manager;
GO

CREATE ROLE Facilities\_Manager;
GO

CREATE ROLE Manager;
GO

Now that we have our database roles, we need our security mapping table and we need it to be populated with some rows to match up the departments with the right database roles:

-- Create the security mapping table and populate it so we can enforce row-level security
CREATE TABLE dbo.SecurityMap (
DepartmentID INT NOT NULL,
RoleName NVARCHAR(255),
CONSTRAINT PK\_SecurityMap PRIMARY KEY CLUSTERED (DepartmentID, RoleName)
);
GO

INSERT INTO dbo.SecurityMap (DepartmentID, RoleName) VALUES (1, 'IT\_Manager');
INSERT INTO dbo.SecurityMap (DepartmentID, RoleName) VALUES (3, 'Marketing\_Manager');
INSERT INTO dbo.SecurityMap (DepartmentID, RoleName) VALUES (4, 'Facilities\_Manager');
GO

Finally, we need the view that uses the SecurityMap table to do the filtering:

-- Create a view which uses row-level security to filter the roles
-- Note the OR IS\_MEMBER('HR\_User') = 1 as well as the OR IS\_MEMBER('db\_owner') = 1
-- as part of the WHERE clause. If a user is a member of either of these roles,
-- then all the rows will be returned.
-- Otherwise, the only way a user will get a row back is if he/she is mapped to the
-- appropriate manager role for that department based on the SecurityMap table.
CREATE VIEW dbo.DepartmentEmployees
AS
SELECT Emp.FirstName, Emp.MiddleName, Emp.SurName, Dept.DepartmentName
FROM dbo.Department Dept
INNER JOIN dbo.Employee Emp
ON Dept.DepartmentID = Emp.DepartmentID
LEFT JOIN dbo.SecurityMap
ON Emp.DepartmentID = dbo.SecurityMap.DepartmentID
WHERE IS\_MEMBER(dbo.SecurityMap.RoleName) = 1
OR IS\_MEMBER('HR\_User') = 1
OR IS\_MEMBER('db\_owner') = 1;
GO

GRANT SELECT ON dbo.DepartmentEmployees TO Manager;
GRANT SELECT ON dbo.DepartmentEmployees TO HR\_User;

Note that the "magic" is being worked by the IS\_MEMBER() function. If the user is a member of the role which matches up to the department based on the SecurityMap table, the user sees the row containing the employee information. If not, the only way the user sees the row is if the user is a member of the HR\_User role or is a member of the db\_owner role, both roles of which see all the data. If all three of those are false for a given row, then the user is not able to see the data on the employee. We can run a series of tests by creating users who are members of the various roles to verify row-level security is working:

-- Query view as DBA
SELECT \* FROM dbo.DepartmentEmployees;

-- Query as HR\_User
CREATE USER Nancy WITHOUT LOGIN;
GO

EXEC sp\_addrolemember @membername = 'Nancy', @rolename = 'HR\_User';
GO

EXECUTE AS USER = 'Nancy';
GO

SELECT \* FROM dbo.DepartmentEmployees;
GO

REVERT;
GO

-- Query as Marketing Manager
CREATE USER Mark WITHOUT LOGIN;
GO

EXEC sp\_addrolemember @membername = 'Mark', @rolename = 'Marketing\_Manager';
EXEC sp\_addrolemember @membername = 'Mark', @rolename = 'Manager';
GO

EXECUTE AS USER = 'Mark';
GO

SELECT \* FROM dbo.DepartmentEmployees;
GO

REVERT;
GO

-- Query as IT Manager
CREATE USER Sydney WITHOUT LOGIN;
GO

EXEC sp\_addrolemember @membername = 'Sydney', @rolename = 'IT\_Manager';
EXEC sp\_addrolemember @membername = 'Sydney', @rolename = 'Manager';
GO

EXECUTE AS USER = 'Sydney';
GO

SELECT \* FROM dbo.DepartmentEmployees;
GO

REVERT;
GO

-- Query as Facilities Manager
CREATE USER Leonard WITHOUT LOGIN;
GO

EXEC sp\_addrolemember @membername = 'Leonard', @rolename = 'Facilities\_Manager';
EXEC sp\_addrolemember @membername = 'Leonard', @rolename = 'Manager';
GO

EXECUTE AS USER = 'Leonard';
GO

SELECT \* FROM dbo.DepartmentEmployees;
GO

REVERT;
GO

And there you have a simple way of tying row-level security to the database roles. If the Windows groups (or even the Windows users) are placed in the appropriate database roles, when they query the DepartmentEmployees view, they'll get the filtered results as appropriate for their department.

**Next Steps**

* Try implementing a solution like this to limit the data that particular users can access without having to write code for each scenario.
* Setup this up using Windows groups or Windows users to use Windows Authentication