Exercise - Manage authorization through column and row level security

9 minutes

In this exercise, examples are shown how you can manage authorization through column and row level security.

An example of column level security

The following example shows how to restrict `TestUser` from accessing the `SSN` column of the `Membership` table:

Create `Membership` table with SSN column used to store social security numbers:

SQL

```
CREATE TABLE Membership
  (MemberID int IDENTITY,
   FirstName varchar(100) NULL,
   SSN char(9) NOT NULL,
   LastName varchar(100) NOT NULL,
   Phone varchar(12) NULL,
   Email varchar(100) NULL);
```

Allow `TestUser` to access all columns except for the SSN column, which has the sensitive data:

SQL

```
GRANT SELECT ON Membership(MemberID, FirstName, LastName, Phone, Email) TO
TestUser;
```

Queries executed as `TestUser` will fail if they include the SSN column:

SQL

```
SELECT * FROM Membership;

-- Msg 230, Level 14, State 1, Line 12
-- The SELECT permission was denied on the column 'SSN' of the object 'Mem-
bership', database 'CLS_TestDW', schema 'dbo'.
```

An example of row level security

This scenario gives you an example for row level security on an Azure Synapse external table.

This short example creates three users and an external table with six rows. It then creates an inline table-valued function and a security policy for the external table. The example shows how select statements are filtered for the various users.

Prerequisites

- You must have a SQL pool. See Create a Synapse SQL pool
- The server hosting your SQL pool must be registered with AAD and you must have an Azure storage account with Storage Blog Contributor permissions. Follow the steps here.
- Create a file system for your Azure Storage account. Use Storage Explorer to view your storage account. Right click on containers and select *Create file system*.

Once you have the prerequisites in place, create three user accounts that will demonstrate different access capabilities.

SQL

```
--run in master
CREATE LOGIN Manager WITH PASSWORD = '<user_password>'
GO
CREATE LOGIN Sales1 WITH PASSWORD = '<user_password>'
GO
CREATE LOGIN Sales2 WITH PASSWORD = '<user_password>'
GO

--run in master and your SQL pool database
CREATE USER Manager FOR LOGIN Manager;
CREATE USER Sales1  FOR LOGIN Sales1;
CREATE USER Sales2  FOR LOGIN Sales2 ;
```

Create a table to hold data.

SQL

```
CREATE TABLE Sales
    (
    OrderID int,
    SalesRep sysname,
    Product varchar(10),
    Qty int
    );
```

Populate the table with six rows of data, showing three orders for each sales representative.

SQL

```
INSERT INTO Sales VALUES (1, 'Sales1', 'Valve', 5);
INSERT INTO Sales VALUES (2, 'Sales1', 'Wheel', 2);
INSERT INTO Sales VALUES (3, 'Sales1', 'Valve', 4);
INSERT INTO Sales VALUES (4, 'Sales2', 'Bracket', 2);
INSERT INTO Sales VALUES (5, 'Sales2', 'Wheel', 5);
INSERT INTO Sales VALUES (6, 'Sales2', 'Seat', 5);
-- View the 6 rows in the table
SELECT * FROM Sales;
```

Create an Azure Synapse external table from the Sales table you just created.

SQL

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<user_password>';

CREATE DATABASE SCOPED CREDENTIAL msi_cred WITH IDENTITY = 'Managed Service
Identity';

CREATE EXTERNAL DATA SOURCE ext_datasource_with_abfss WITH (TYPE = hadoop,
LOCATION = 'abfss://<file_system_name@storage_account>.dfs.core.win-
dows.net', CREDENTIAL = msi_cred);

CREATE EXTERNAL FILE FORMAT MSIFormat  WITH (FORMAT_TYPE=DELIMITEDTEXT);

CREATE EXTERNAL TABLE Sales_ext WITH (LOCATION='<your_table_name>', DATA_-
SOURCE=ext_datasource_with_abfss, FILE_FORMAT=MSIFormat, REJECT_TYPE=Per-
centage, REJECT_SAMPLE_VALUE=100, REJECT_VALUE=100)
AS SELECT * FROM sales;
```

Grant SELECT for the three users on the external table Sales_ext that you created.

SQL

```
GRANT SELECT ON Sales_ext TO Sales1;
GRANT SELECT ON Sales_ext TO Sales2;
GRANT SELECT ON Sales_ext TO Manager;
```

Create a new schema, and an inline table-valued function, you may have completed this in example A. The function returns 1 when a row in the SalesRep column is the same as the user executing the query (`@SalesRep = USER_NAME()`) or if the user executing the query is the Manager user (`USER_NAME() = 'Manager'`).

SQL

```
CREATE SCHEMA Security;
GO

CREATE FUNCTION Security.fn_securitypredicate(@SalesRep AS sysname)
    RETURNS TABLE
WITH SCHEMABINDING
AS
    RETURN SELECT 1 AS fn_securitypredicate_result
WHERE @SalesRep = USER_NAME() OR USER_NAME() = 'Manager';
```

Create a security policy on your external table using the inline table-valued function as a filter predicate. The state must be set to ON to enable the policy.

SQL

```
CREATE SECURITY POLICY SalesFilter_ext
ADD FILTER PREDICATE Security.fn_securitypredicate(SalesRep)
ON dbo.Sales_ext
WITH (STATE = ON);
```

Now test the filtering predicate, by selecting from the Sales_ext external table. Sign in as each user, Sales1, Sales2, and manager. Run the following command as each user.

SQL

```
SELECT * FROM Sales_ext;
```

The Manager should see all six rows. The Sales1 and Sales2 users should only see their sales.

Alter the security policy to disable the policy.

SQL

```
ALTER SECURITY POLICY SalesFilter_ext
WITH (STATE = OFF);
```

Now the Sales1 and Sales2 users can see all six rows.

Connect to the Azure Synapse database to clean up resources

SQL

```
DROP USER Sales1;
DROP USER Sales2;
DROP USER Manager;

DROP SECURITY POLICY SalesFilter_ext;
```

```
DROP TABLE Sales;
DROP EXTERNAL TABLE Sales_ext;
DROP EXTERNAL DATA SOURCE ext_datasource_with_abfss ;
DROP EXTERNAL FILE FORMAT MSIFormat;
DROP DATABASE SCOPED CREDENTIAL msi_cred;
DROP MASTER KEY;
```

Connect to logical master to clean up resources.

SQL

```
DROP LOGIN Sales1;
DROP LOGIN Sales2;
DROP LOGIN Manager;
```
