**Topic**: Measure the number of Influenza cases in U.S using Data management and masking on Azure

Problem Statement: Although most organizations have stringent security controls in place to protect production data in storage or in business use, sometimes that same data has been used for operations that are less secure. The issue is often compounded if these operations are outsourced and the organization has less control over the environment. In the wake of [compliance](http://searchdatamanagement.techtarget.com/definition/compliance) legislation, most organizations are no longer comfortable exposing real data unnecessarily. In data masking, the format of data remains the same; only the values are changed.

In this example we are going to demonstrate how to mask the patient data and make it available for the analytics. We are going to measure the number of Influenza cases in 2018 in all the United States without comprising any of the patient PHI.

Overview of technology: Azure dynamic data masking is Microsoft’s cloud based platform that limits sensitive data exposure by masking it to the non-privileged users.

Overview of steps:

1. Prepare a clean Patient data set with Influenza cases and save a CSV
2. Create a SQL server and a sql database
3. Create tables and multiple users with access to the newly created database
4. Create azure storage account and container to hold your source data files
5. Create Azure data factory to load the data from flat file into the SQL database
6. Mask the data on the sql server for the required users using Azure Data Masking
7. Connect to Sql Server DB using Tableau and import all the required tables to measure the Influenza cases
8. Build a map report to show the number of Influenza cases in different states of U.S.

Data set obtained at: <https://www.challenge.gov/challenge/patient-matching-algorithm-challenge/>

Hardware/OS: Intel Core i5-5300U CPU 2.30 GHZ, 12 GB RAM, 64 bit Windows 7 operating system

Software used: Azure, Visual Studio 2017, C#, SQL Server Management studio 2017, Tableau 10.5

Lessons learnt: Only dynamic data masking is possible/available. This doesn’t change the base data but just changes the view of the data based on the user access. This might be an issue in some cases where you need to mask the base data and not change the view of the data

Pros: Ease of design and implementation.

* No external tool/algorithm required for dynamic data masking.
* No constraint violation since underlying data is not changed and only the data view is changed
* Simplicity of Microsoft GUI
* Data Masking is not reversible for unauthorized users.

Cons:

* Data cannot be masked using custom dictionaries
* Limited customization available out of the box for dynamic data masking

YouTube Links:

* 2 Min: <https://youtu.be/JVKM5vFOiY8>
* 15 Min: <https://youtu.be/2ye9KoqC4Xo>

# Prerequisites:

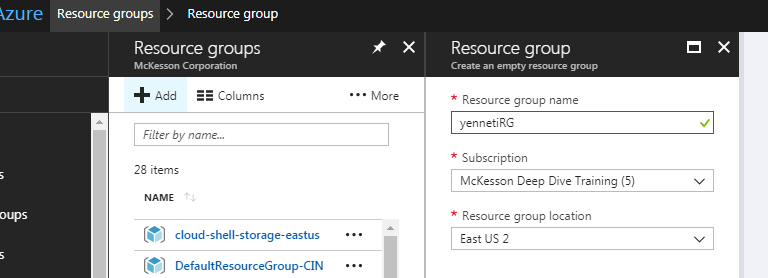
1. MS SQL server Management studio 2017
2. Windows 7
3. MS Word
4. Dataset
5. Visual Studio
6. Tableau 10.5

# Identify Prepare the Data Set:

1. Import the data set from <https://www.challenge.gov/challenge/patient-matching-algorithm-challenge/>
2. Make sure the data is clean .
3. Save the dataset as text files. I have two text files here Patient.txt and Provider.Txt
4. I am using only 50K records for this exercise.
5. Patient table has a provider Id .

## Create a Resource Group in Azure Portal:

Select Resource group-> Add and give all the details for new resource group.



Create a SQL Server and SQL Database:

Select new -> SQL Database

I gave a name for your Data base as **yennetiDB**

Select the Subscription.

Select the resource group that you created in the previous step.

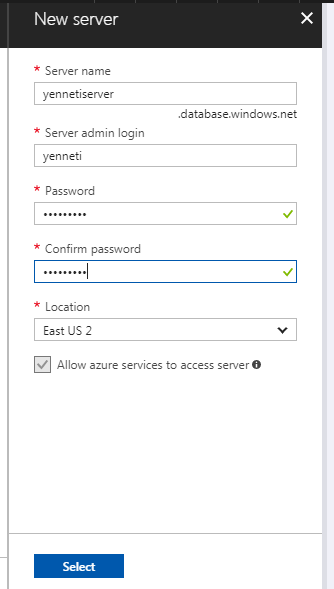
Select ‘**Blank Database’** as the source

In the server tab -> Create a new server->

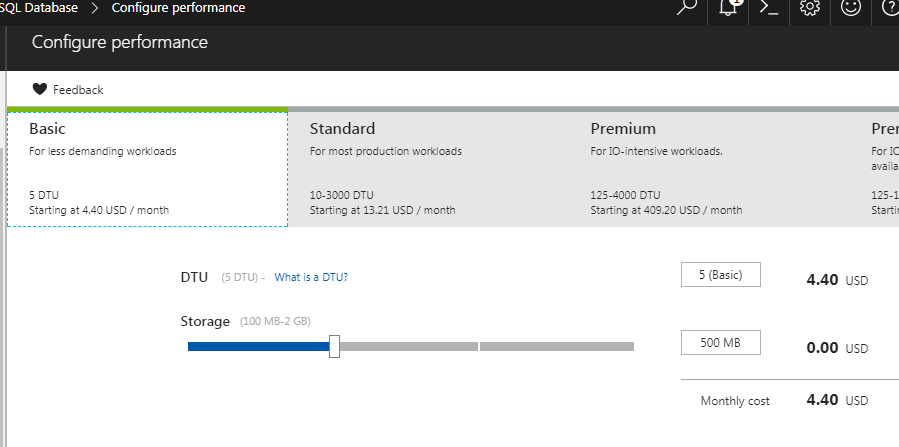
Server name : **Yennetiserver**

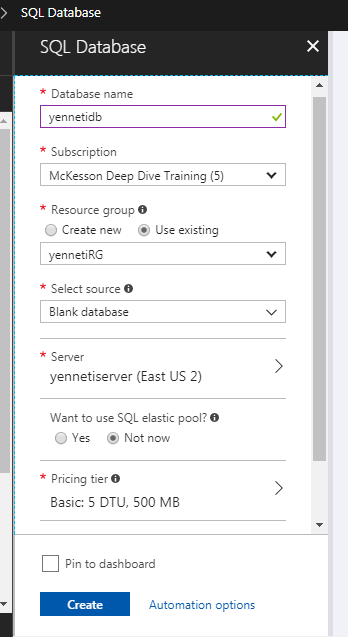
Server admin login: **Yenneti**

Location: **East US 2**



**Make sure you select and change the pricing tier to Basic . The default pricing comes with Standard**



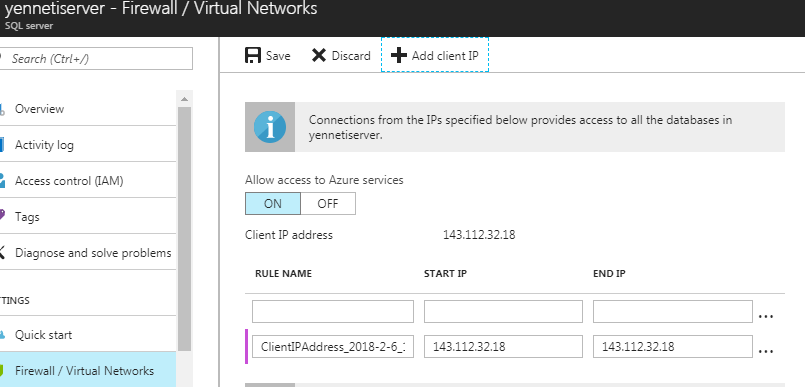


## Add a firewall:

We need to add a firewall to the server.

Open the **yennetirg** Resource Group

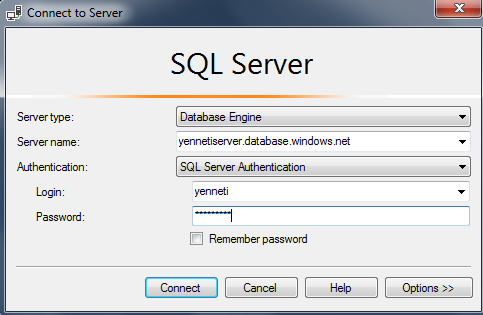
Open the yennetiserver ->settings->firewall->Add client IP->save



Create the tables and users in the Database:

Open the SQL Server Management Studio and login to the server using the credentials you gave while creating the server.

Server name would be yourservername.database.windows.net . If you forget the admin any time

Goto server->properties

Go to the ‘yennetidb’ database you created in the prior step and create the required tables.

(Right click on the db and say new query)

CREATE TABLE PATIENT

(

ENTERPRISEID VARCHAR(50 ),

LAST\_NAME VARCHAR(50 ),

FIRST\_NAME VARCHAR(50 ),

DOB date,

GENDER VARCHAR(50 ),

SSN VARCHAR(50 ),

ADDRESS1 VARCHAR(50 ),

ZIP VARCHAR(50 ),

MRN VARCHAR(50 ),

CITY VARCHAR(50 ),

STATE VARCHAR(50 ),

PHONE VARCHAR(50 ),

EMAIL VARCHAR(100 ),

PRVDR\_ID VARCHAR(50)

)

CREATE TABLE PROVIDER

(

ID VARCHAR(50 ),

PRVDR\_FIRST\_NAME VARCHAR(255 ),

PRVDR\_LAST\_NAME VARCHAR(255 ),

PRVDR\_DEA VARCHAR(255 ),

PRVDR\_NPI VARCHAR(255 ),

PRIMARY\_SPECIALITY VARCHAR(255 ),

PRIMARY\_STATE\_LICENSE\_NO VARCHAR(100 ),

SECONDARY\_STATE\_LICENSE\_NO VARCHAR(100 )

)

create Table Diag

(

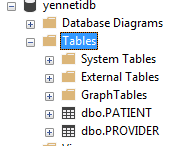
patient varchar(50),

icd varchar(50),

Diag varchar(50)

)

If you refresh the table list you should be seeing the new tables



## Create users:

We need to create couple of users to show different data privilages.

Go the master database and create the users. After creating create the users to the yennetidb

CREATE LOGIN Azurereadonly WITH password='password@2'

GO

CREATE USER [Azurereadonly] FOR LOGIN [Azurereadonly] WITH DEFAULT\_SCHEMA=[dbo]

GO

CREATE LOGIN Azurereadwrite1 WITH password='password@2'

GO

CREATE USER [Azurereadwrite1] FOR LOGIN [Azurereadwrite1] WITH DEFAULT\_SCHEMA=[dbo]

GO

EXEC sp\_addrolemember 'db\_datareader', 'Azurereadonly';

GO

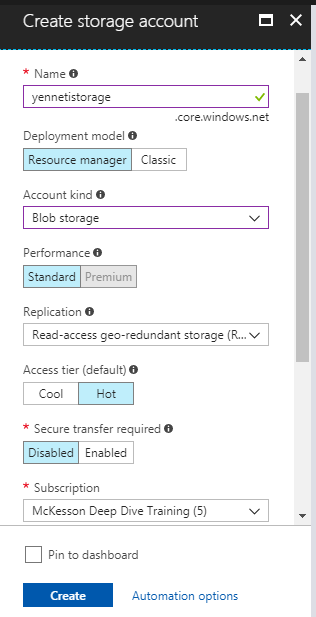
EXEC sp\_addrolemember 'db\_datawriter', 'Azurereadwrite1';

GO

# Create a Storage account and Container :

Go to Storage account -> Add

And enter all the details. Here I selected ‘Blob Storage’ as Account Kind

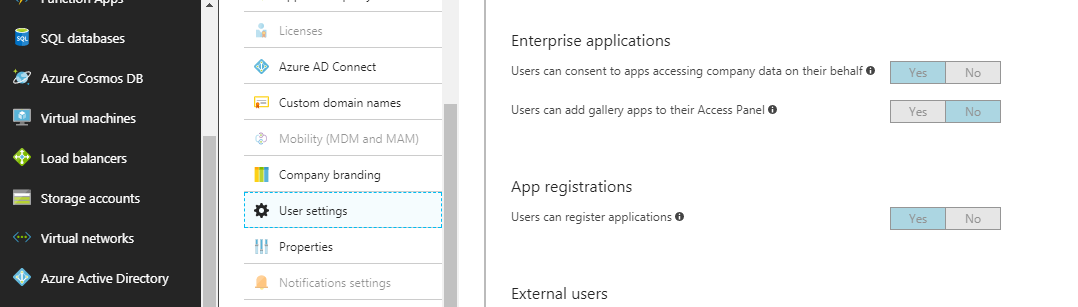


Go to yennetistorage->settings->Access Keys

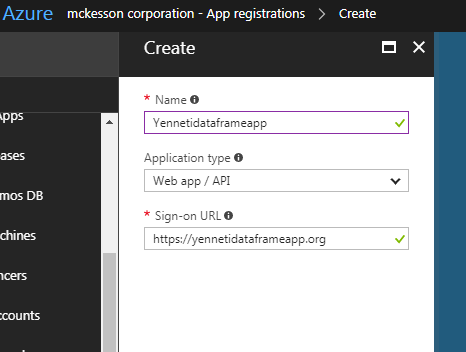
And copy the key1 in a notepad. You would be needing this to create the Datafactory.

## Create a Data Frame:

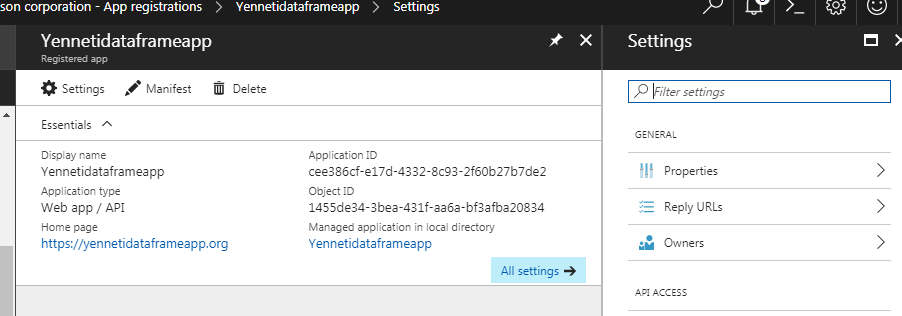
Go to Azure active directory and make sure the app registration is set to yes in user settings



Go to app registration->create

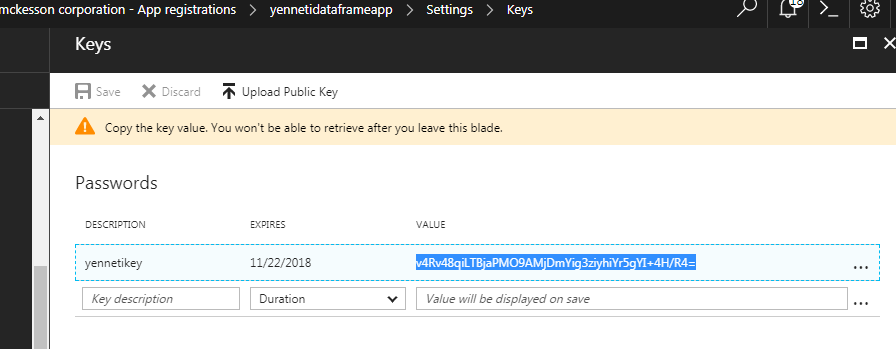


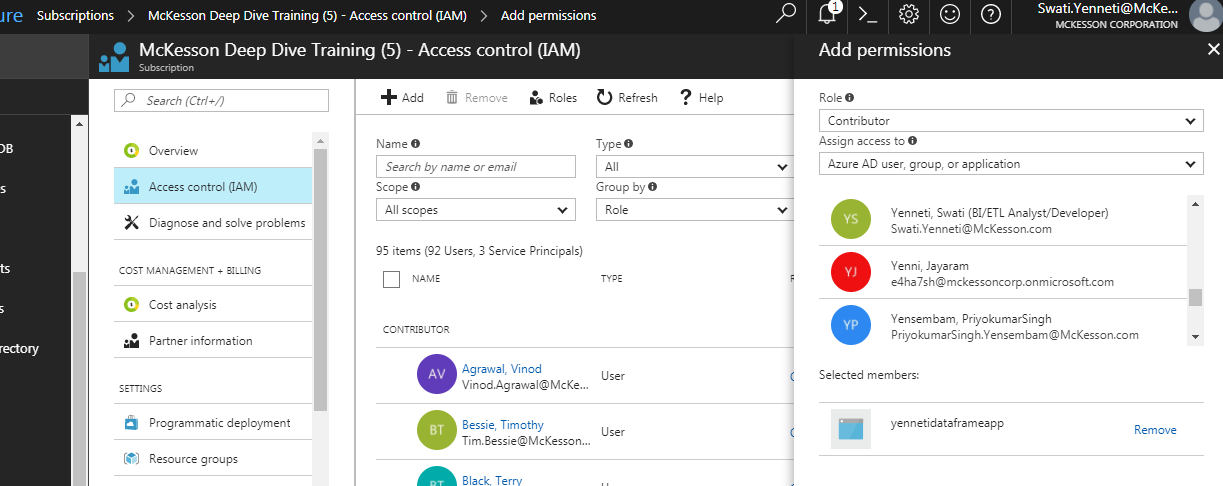
yennetidataframeapp

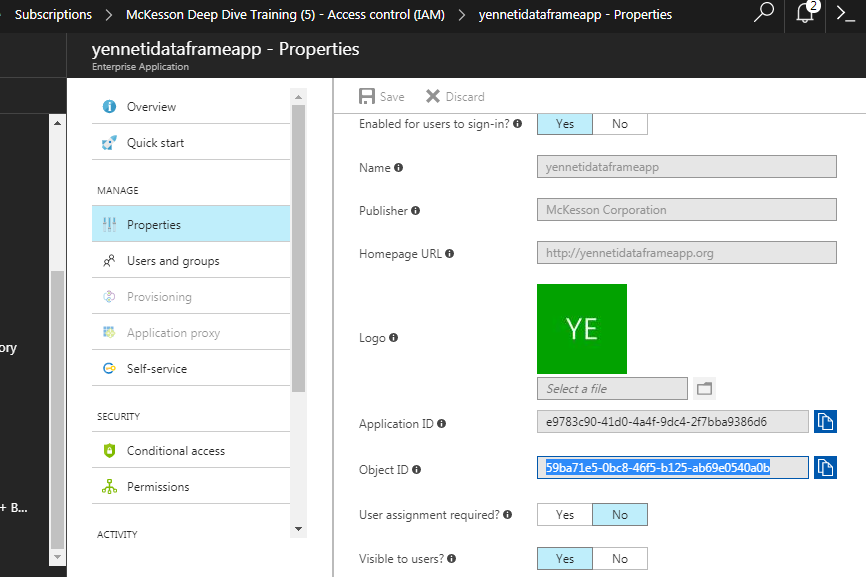


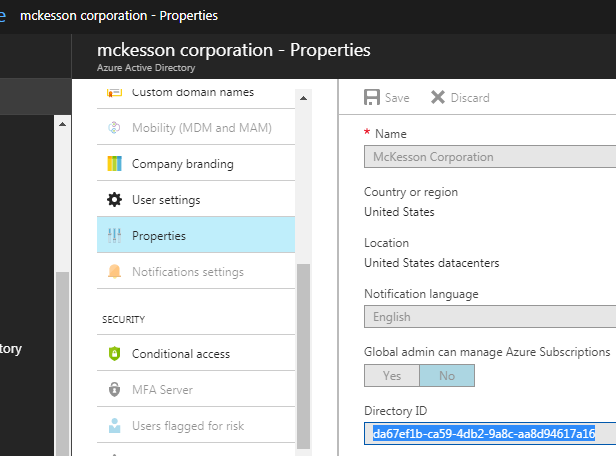
Click All setting and select keys :

In the page enter the key description and select save to generate the key



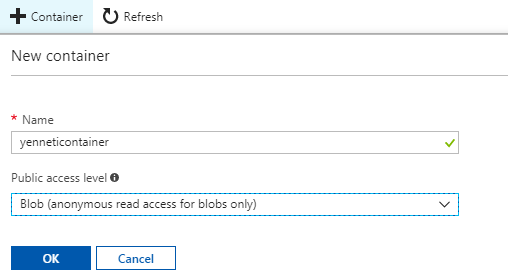




Directory ID:

Go to yennetistorage->settings->containers

Select ‘+container’ to add a new container. Create a new container with Blob as Public level access



Open the container and select upload to upload your flat files.

Make sure all your files are in the container

## Create DataFactory:

## Create a Visual Studio project

Using Visual Studio 2013/2015/2017, create a C# .NET console application.

1. Launch Visual Studio.
2. Click File, point to New, and click Project.
3. Select Visual C# -> Console App (.NET Framework) from the list of project types on the right. .NET version 4.5.2 or above is required.
4. Enter yennetiapp for the Name.
5. Click OK to create the project.

## Install NuGet packages

1. Click **Tools** -> **NuGet Package Manager** -> **Package Manager Console**.
2. In the **Package Manager Console**, run the following commands to install packages:

Copy

Install-Package Microsoft.Azure.Management.DataFactory -Prerelease

Install-Package Microsoft.Azure.Management.ResourceManager -Prerelease

Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory

## Create a data factory client

1. Open Program.cs, include the following statements to add references to namespaces.

C#Copy

using System;

using System.Collections.Generic;

using System.Linq;

using Microsoft.Rest;

using Microsoft.Azure.Management.ResourceManager;

using Microsoft.Azure.Management.DataFactory;

using Microsoft.Azure.Management.DataFactory.Models;

using Microsoft.IdentityModel.Clients.ActiveDirectory;

1. Add the following code to the Main method that sets the variables. Replace the place-holders with your own values. Currently, Data Factory V2 allows you to create data factories only in the East US, East US2, and West Europe regions. The data stores (Azure Storage, Azure SQL Database, etc.) and computes (HDInsight, etc.) used by data factory can be in other regions.

// Set variables

string tenantID = "<your tenant ID>";

string applicationId = "<your application ID>";

string authenticationKey = "<your authentication key for the application>";

string subscriptionId = "<your subscription ID where the data factory resides>";

string resourceGroup = "<your resource group where the data factory resides>";

string region = "East US 2";

string dataFactoryName = "<specify the name of data factory to create. It must be globally unique.>";

string storageAccount = "<your storage account name to copy data>";

string storageKey = "<your storage account key>";

// specify the container and input folder from which all files need to be copied to the output folder.

string inputBlobPath = "<the path to existing blob(s) to copy data from, e.g. containername/foldername>";

//specify the contains and output folder where the files are copied

string outputBlobPath = "<the blob path to copy data to, e.g. containername/foldername>";

string storageLinkedServiceName = "AzureStorageLinkedService"; // name of the Azure Storage linked service

string blobDatasetName = "BlobDataset"; // name of the blob dataset

string pipelineName = "Adfv2QuickStartPipeline"; // name of the pipeline

Add the following code to the **Main** method that creates an instance of **DataFactoryManagementClient** class. You use this object to create a data factory, a linked service, datasets, and a pipeline. You also use this object to monitor the pipeline run details.

// Authenticate and create a data factory management client

var context = new AuthenticationContext("https://login.windows.net/" + tenantID);

ClientCredential cc = new ClientCredential(applicationId, authenticationKey);

AuthenticationResult result = context.AcquireTokenAsync("https://management.azure.com/", cc).Result;

ServiceClientCredentials cred = new TokenCredentials(result.AccessToken);

var client = new DataFactoryManagementClient(cred) { SubscriptionId = subscriptionId };

Add the following code to the **Main** method that creates a **data factory**.

// Create a data factory

Console.WriteLine("Creating data factory " + dataFactoryName + "...");

Factory dataFactory = new Factory

{

Location = region,

Identity = new FactoryIdentity()

};

client.Factories.CreateOrUpdate(resourceGroup, dataFactoryName, dataFactory);

Console.WriteLine(SafeJsonConvert.SerializeObject(dataFactory, client.SerializationSettings));

while (client.Factories.Get(resourceGroup, dataFactoryName).ProvisioningState == "PendingCreation")

{

System.Threading.Thread.Sleep(1000);

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Microsoft.Rest;

using Microsoft.Azure.Management.ResourceManager;

using Microsoft.Azure.Management.DataFactory;

using Microsoft.Azure.Management.DataFactory.Models;

using Microsoft.IdentityModel.Clients.ActiveDirectory;

namespace ConsoleApp3

{

class Program

{

static void Main(string[] args)

{

/// Set variables

string tenantID = "da67ef1b-ca59-4db2-9a8c-aa8d94617a16";

string applicationId = "99eb9b37-58dd-416d-98b1-750d7987e8f2";

string authenticationKey = "MSPUrCZVyoutsLj5eeiTAm/LBAKWywXG0Qltdll2g2I=";

string subscriptionId = "4d05309c-0709-4c19-8ea1-bf8eef09ef16";

string resourceGroup = "yennetirg";

string region = "East US 2";

string dataFactoryName = "yennetidatafactory";

// Specify the source Azure Blob information

string storageAccount = "yennetistorage";

string storageKey = "uz0Wg7Ndm2Mco2b7nKuzFGOh0A77wilzwAcNXa4Z4lOVTHykGiDdSZ0NJa31A9XVnu7ej7a3Zi6Auq3ncP7BCQ==";

string inputBlobPath = "yenneticontainer/";

string inputBlobName = "PATIENT.csv";

// Specify the sink Azure SQL Database information

string azureSqlConnString = "Server=tcp:yennetiserver.database.windows.net,1433;Database=yennetidb;User ID=yenneti@yennetiserver.database.windows.net;Password=Password1;Trusted\_Connection=False;Encrypt=True;Connection Timeout = 30";

string azureSqlTableName = "dbo.patient";

string storageLinkedServiceName = "AzureStorageLinkedService";

string sqlDbLinkedServiceName = "AzureSqlDbLinkedService";

string blobDatasetName = "BlobDataset";

string sqlDatasetName = "SqlDataset";

string pipelineName = "BlobToSqlCopy";

// Authenticate and create a data factory management client

var context = new AuthenticationContext("https://login.windows.net/" + tenantID);

ClientCredential cc = new ClientCredential(applicationId, authenticationKey);

AuthenticationResult result = context.AcquireTokenAsync("https://management.azure.com/", cc).Result;

ServiceClientCredentials cred = new TokenCredentials(result.AccessToken);

var client = new DataFactoryManagementClient(cred) { SubscriptionId = subscriptionId };

// Create a data factory

Console.WriteLine("Creating a data factory " + dataFactoryName + "...");

Factory dataFactory = new Factory

{

Location = region,

Identity = new FactoryIdentity()

};

client.Factories.CreateOrUpdate(resourceGroup, dataFactoryName, dataFactory);

Console.WriteLine(SafeJsonConvert.SerializeObject(dataFactory, client.SerializationSettings));

while (client.Factories.Get(resourceGroup, dataFactoryName).ProvisioningState == "PendingCreation")

{

System.Threading.Thread.Sleep(1000);

}

// Create an Azure Storage linked service

Console.WriteLine("Creating linked service " + storageLinkedServiceName + "...");

LinkedServiceResource storageLinkedService = new LinkedServiceResource(

new AzureStorageLinkedService

{

ConnectionString = new SecureString("DefaultEndpointsProtocol=https;AccountName=" + storageAccount + ";AccountKey=" + storageKey)

}

);

client.LinkedServices.CreateOrUpdate(resourceGroup, dataFactoryName, storageLinkedServiceName, storageLinkedService);

Console.WriteLine(SafeJsonConvert.SerializeObject(storageLinkedService, client.SerializationSettings));

// Create an Azure SQL Database linked service

Console.WriteLine("Creating linked service " + sqlDbLinkedServiceName + "...");

LinkedServiceResource sqlDbLinkedService = new LinkedServiceResource(

new AzureSqlDatabaseLinkedService

{

ConnectionString = new SecureString(azureSqlConnString)

}

);

client.LinkedServices.CreateOrUpdate(resourceGroup, dataFactoryName, sqlDbLinkedServiceName, sqlDbLinkedService);

Console.WriteLine(SafeJsonConvert.SerializeObject(sqlDbLinkedService, client.SerializationSettings));

// Create a Azure Blob dataset

Console.WriteLine("Creating dataset " + blobDatasetName + "...");

DatasetResource blobDataset = new DatasetResource(

new AzureBlobDataset

{

LinkedServiceName = new LinkedServiceReference

{

ReferenceName = storageLinkedServiceName

},

FolderPath = inputBlobPath,

FileName = inputBlobName,

Format = new TextFormat { ColumnDelimiter = "" +"|" },

Structure = new List<DatasetDataElement>

{

/\*

new DatasetDataElement

{

Name = "ID",

Type = "String"

},

new DatasetDataElement

{

Name = "PRVDR\_FIRST\_NAME",

Type = "String"

},

new DatasetDataElement

{

Name = "PRVDR\_LAST\_NAME",

Type = "String"

},

new DatasetDataElement

{

Name = "PRVDR\_DEA",

Type = "String"

},

new DatasetDataElement

{

Name = "PRVDR\_NPI",

Type = "String"

},

new DatasetDataElement

{

Name = "PRIMARY\_SPECIALITY",

Type = "String"

},

new DatasetDataElement

{

Name = "PRIMARY\_STATE\_LICENSE\_NO",

Type = "String"

},

new DatasetDataElement

{

Name = "SECONDARY\_STATE\_LICENSE\_NO",

Type = "String"

}

\*/

new DatasetDataElement

{

Name = "ENTERPRISEID",

Type = "String"

},

new DatasetDataElement

{

Name = "LAST\_NAME",

Type = "String"

},

new DatasetDataElement

{

Name = "FIRST\_NAME",

Type = "String"

},

new DatasetDataElement

{

Name = "DOB",

Type = "Date"

},

new DatasetDataElement

{

Name = "GENDER",

Type = "String"

},

new DatasetDataElement

{

Name = "SSN",

Type = "String"

},

new DatasetDataElement

{

Name = "ADDRESS1",

Type = "String"

},

new DatasetDataElement

{

Name = "ZIP",

Type = "String"

},

new DatasetDataElement

{

Name = "MRN",

Type = "String"

},

new DatasetDataElement

{

Name = "CITY",

Type = "String"

},

new DatasetDataElement

{

Name = "STATE",

Type = "String"

},

new DatasetDataElement

{

Name = "PHONE",

Type = "String"

},

new DatasetDataElement

{

Name = "EMAIL",

Type = "String"

},

new DatasetDataElement

{

Name = "PRVDR\_ID",

Type = "String"

}

/\*

new DatasetDataElement

{

Name = "patient",

Type = "String"

},

new DatasetDataElement

{

Name = "icd",

Type = "String"

},

new DatasetDataElement

{

Name = "Diag",

Type = "String"

}

\*/

}

}

);

client.Datasets.CreateOrUpdate(resourceGroup, dataFactoryName, blobDatasetName, blobDataset);

Console.WriteLine(SafeJsonConvert.SerializeObject(blobDataset, client.SerializationSettings));

// Create a Azure SQL Database dataset

Console.WriteLine("Creating dataset " + sqlDatasetName + "...");

DatasetResource sqlDataset = new DatasetResource(

new AzureSqlTableDataset

{

LinkedServiceName = new LinkedServiceReference

{

ReferenceName = sqlDbLinkedServiceName

},

TableName = azureSqlTableName

}

);

client.Datasets.CreateOrUpdate(resourceGroup, dataFactoryName, sqlDatasetName, sqlDataset);

Console.WriteLine(SafeJsonConvert.SerializeObject(sqlDataset, client.SerializationSettings));

// Create a pipeline with copy activity

Console.WriteLine("Creating pipeline " + pipelineName + "...");

PipelineResource pipeline = new PipelineResource

{

Activities = new List<Activity>

{

new CopyActivity

{

Name = "CopyFromBlobToSQL",

Inputs = new List<DatasetReference>

{

new DatasetReference()

{

ReferenceName = blobDatasetName

}

},

Outputs = new List<DatasetReference>

{

new DatasetReference

{

ReferenceName = sqlDatasetName

}

},

Source = new BlobSource { },

Sink = new SqlSink { }

}

}

};

client.Pipelines.CreateOrUpdate(resourceGroup, dataFactoryName, pipelineName, pipeline);

Console.WriteLine(SafeJsonConvert.SerializeObject(pipeline, client.SerializationSettings));

// Create a pipeline run

Console.WriteLine("Creating pipeline run...");

CreateRunResponse runResponse = client.Pipelines.CreateRunWithHttpMessagesAsync(resourceGroup, dataFactoryName, pipelineName).Result.Body;

Console.WriteLine("Pipeline run ID: " + runResponse.RunId);

// Monitor the pipeline run

Console.WriteLine("Checking pipeline run status...");

PipelineRun pipelineRun;

while (true)

{

pipelineRun = client.PipelineRuns.Get(resourceGroup, dataFactoryName, runResponse.RunId);

Console.WriteLine("Status: " + pipelineRun.Status);

if (pipelineRun.Status == "InProgress")

System.Threading.Thread.Sleep(15000);

else

break;

}

// Check the copy activity run details

Console.WriteLine("Checking copy activity run details...");

List<ActivityRun> activityRuns = client.ActivityRuns.ListByPipelineRun(

resourceGroup, dataFactoryName, runResponse.RunId, DateTime.UtcNow.AddMinutes(-10), DateTime.UtcNow.AddMinutes(10)).ToList();

if (pipelineRun.Status == "Succeeded")

{

Console.WriteLine(activityRuns.First().Output);

}

else

Console.WriteLine(activityRuns.First().Error);

Console.WriteLine("\nPress any key to exit...");

Console.ReadKey();

}

}

}

;

Run the above code for each table by changing the table/column definition and the source file.

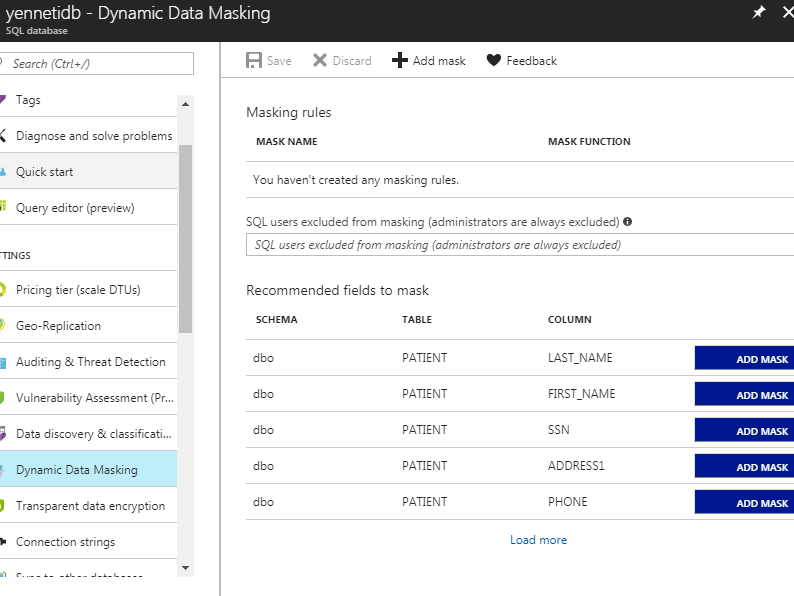
Note: Make sure you create the Data Factory in East US . I wasted several hours not able to find the cause when I was creating in the Central US.

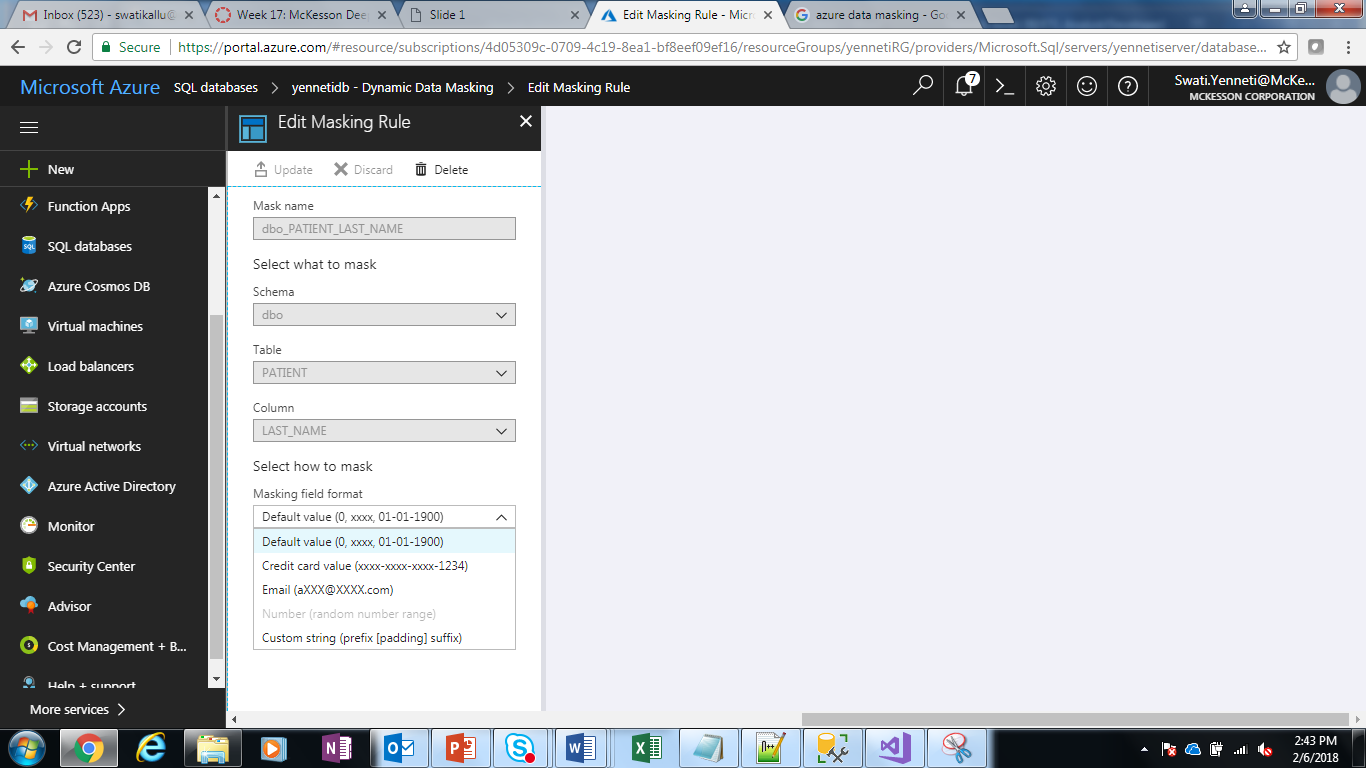
# Masking the database :

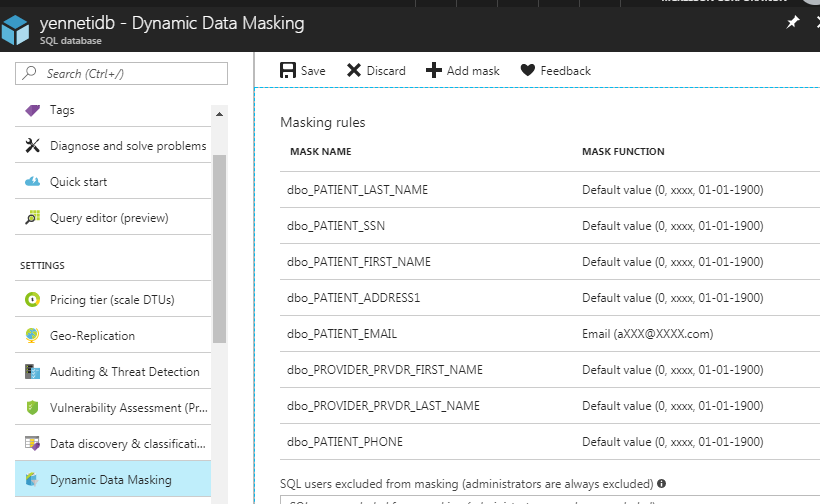
Open the yennetidb -> Settings -> Dynamic Data Masking

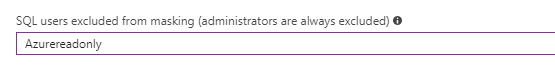
The tool gives the list of recommended columns for the masking .

Once the column is added right click on the column and change the masking format from the available options.









DB administrators can always see unmasked the data.

By default all the users will see the masked data.

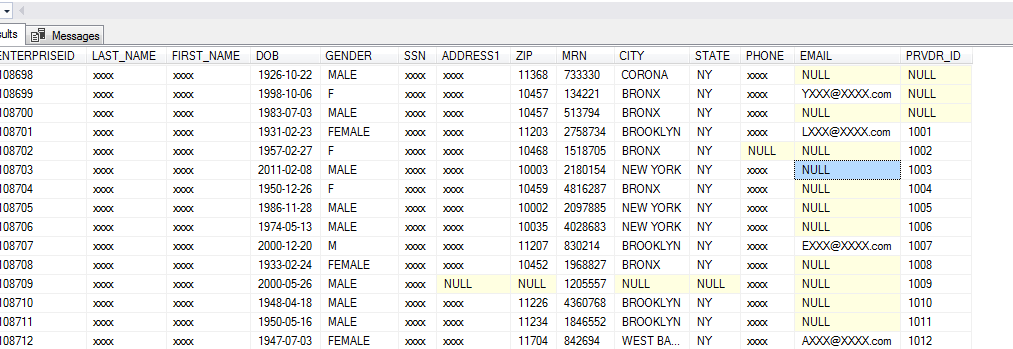
To exclude any user from masking add the users to the box.

SQL users excluded from masking (administrators are always excluded)

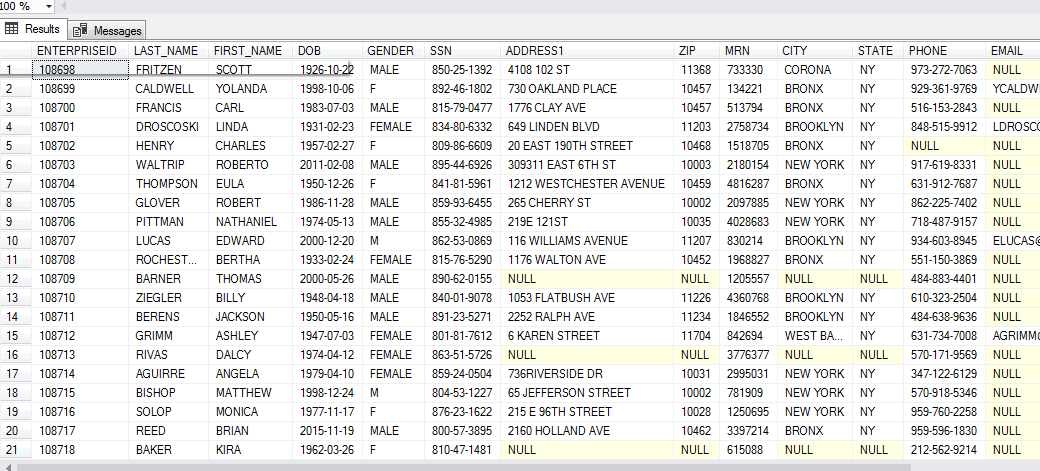
Specify a semicolon-separated list of database SQL users for which data masking will NOT apply. These users will get data results without masking for all database queries.

Click save when after adding all the columns.

I will now login to the db using Azurereadwrite1 login.

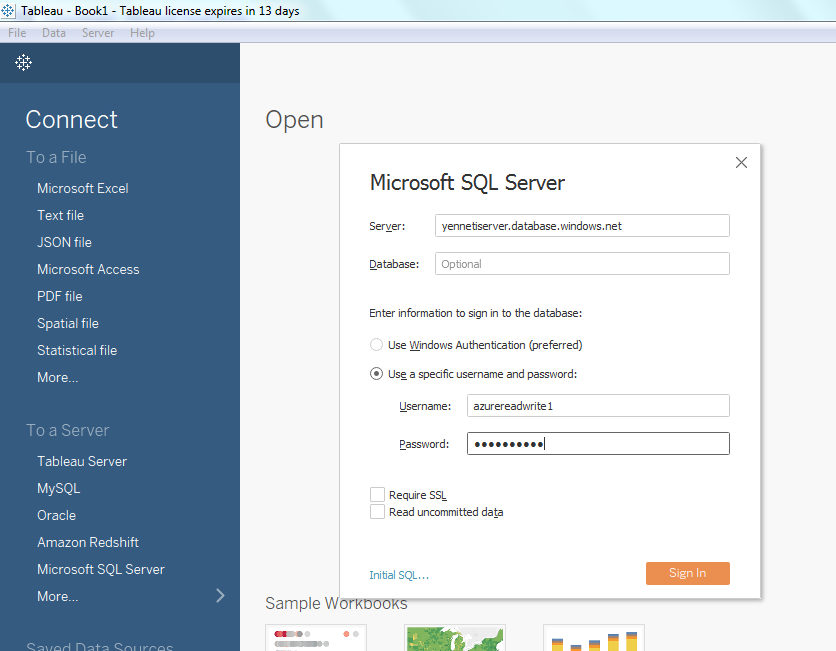


Since Azurereadonly user is excluded from the masking user list he has full view of the data.



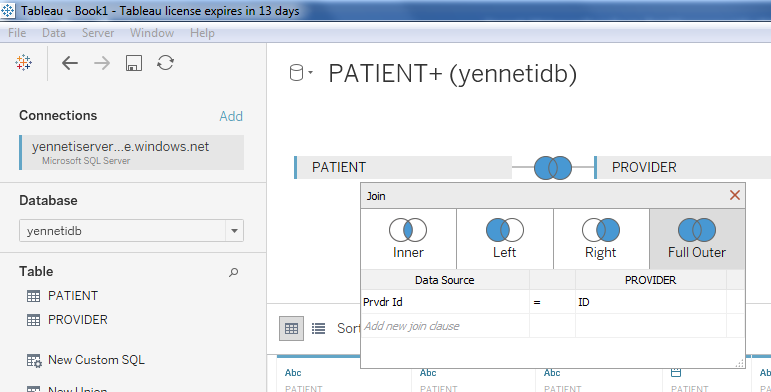
Open Tableau and select Connect->to a server -> MicrosoftSQLserver

I am logging in here as azuremaskeduser

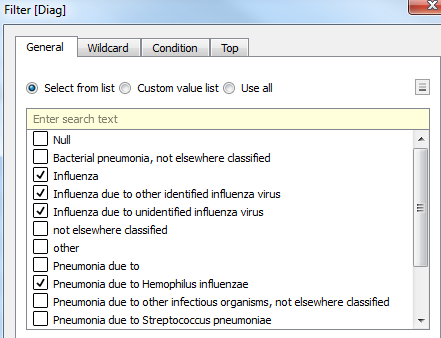


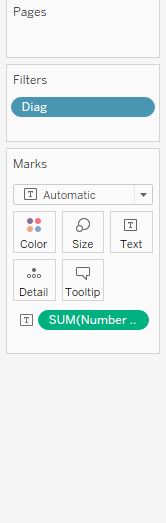
Open yennetidb and add the tables and join the tables patient and provider by provider id

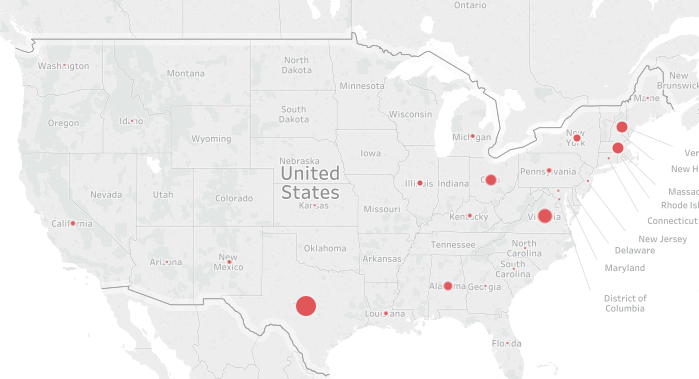
And join the diag and patient by patient and enterpriseid



Add the Diag in the filters .Select only certain values







Report shows that Texas has the maximum number of Influenza cases recorded.

* Two minute (short): <https://youtu.be/JVKM5vFOiY8>
* 15 minutes (long): <https://youtu.be/2ye9KoqC4Xo>
* GitHub Repository with all artifacts: <https://github.com/syenneti/FinalProject.git>

References:

1. <https://docs.microsoft.com/en-us/azure/data-factory/quickstart-create-data-factory-dot-net>
2. <http://deepazure.s3.amazonaws.com/mckesson-labs/lab06/Lab06_Blob_and_DataFactoryPythonv1.pdf>
3. https://docs.microsoft.com/en-us/azure/sql-database/sql-database-dynamic-data-masking-get-started