Sujatha Kestur

02/06/2018

**Topic**: Data management and masking on Azure

**Introduction**

Data masking is a method of creating a structurally similar but inauthentic version of an organization's data. The purpose is to protect the actual data while having a functional substitute for occasions when the real data is not required and to simultaneously maintain the usability of the data. It is used to reduce the unnecessary spread and exposure of sensitive data within or outside an organization.

[Gartner](https://www.gartner.com/doc/3153926/static-dynamic-data-masking-explained) describes it as a technology that **“can dynamically or statistically protect sensitive data by replacing it with fictitious data that looks realistic to prevent data loss in different use cases.” .** Data masking can protect many forms of sensitive data, including (but not limited to):

* Personally identifiable information (PII)
* Protected health information (PHI)
* Payment card information (subject to PCI-DSS regulation)
* Intellectual property (subject to ITAR and EAR regulations)

Data exposure to the right user is essential to properly organize and maintain data processes to meet ongoing information lifecycle needs.

* In the wake of compliance legislation, most organizations are no longer comfortable exposing real data unnecessarily
* Top reasons for including the data masking in the broader data security strategy are
  + Protect non-production data – making copies of your production data onto the development and qa boxes increases the risk of data falling into wrong hands
  + Protect against insider threats
  + For compliance purposes

The **HIPPA** act sets the standard for sensitive data protection. Companies that deal with protected health information (PHI) must have physical, network and process security measures in place and follow them to ensure HIPPA compliance.

**Masking Methodology(Overview)**

The topic of masking brings in a general perspective that **Encryption** and **Masking** are the same. The **Fundamental Difference: For encryption, reversibility is required, and for masking, reversibility is a weakness.**

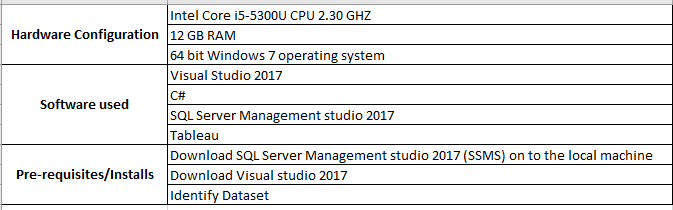
* **Encryption**
  + Involves converting and transforming data into scrambled, often unreadable, cipher-text using non-readable mathematical calculations and algorithms
  + Restoring the message requires a corresponding decryption algorithm and the original encryption key
  + It is widely used to protect files on a local, network or cloud disk drives, network communications, or just web/email traffic protection
* **Persistent Data masking**
  + This technique masks the base data and does not just change the view of the data
  + Scalable data masking software that enables organizations to create safe and secure copies of data by anonymizing information that could threaten the privacy, security, or compliance of critical data
* **Dynamic Data Masking** 
  + Hides data elements that users of certain roles should not see and replaces them with similar-looking fake data
  + Dynamic data masking can transform the data on the fly based on the user role (privileges)
  + It is used to secure real time transactional systems and speeds up data privacy, compliance implementation, and maintenance
  + Data masking does not encrypt information. We can see all data records in their native form and no decryption key is necessary

For the current context of project, we go with **Dynamic data masking**. This is a service provided out of box by Microsoft’s cloud based platform. This functionality limits sensitive data exposure by masking it to the non-privileged users.

**Why Data Masking?**

With umpteen attempts being done in the real world to hack the data and publicize it, it becomes even more important to protect the patient personal information such as Names, date of birth, SSN, address etc. Patient health information and insurance information. The data masking that is offered out of box by Azure only aims at changing the view of the data based on user access rights. The project demonstrates this fact by only showing masked data to unauthorized user. The data view for the authorized user proves that the actual underlying data still remains the same and only the data view changes

**Prerequisites**



**Dataset**

The current project uses three tables: Patient, Provider and Patient\_Metastasis

* **Patient Table** contains all the patient related information such as First name, last name, date of birth, medical record number, race, address, ssn, phone number and email id. This data set was pulled from patient matching challenge dataset which was provided by federal government

<https://www.challenge.gov/challenge/patient-matching-algorithm-challenge/>

* **Provider table** contains all the provider information such as provider name, dea number, NPI, primary specialty
* **Patient Metastasis table** holds the diagnosis information of the patient with the year of diagnosis and a reference to the patient table with the patient id

**Implementation of the project**

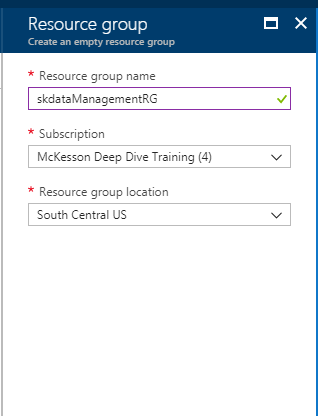
The scope of this project includes moving data from a flat file to the tables on sql server database hosted on Azure and the subsequent masking of the data. The masked data is then shown in the form of reports and graphs using Tablueau, proving that masked view of data does not affect any down stream functionalities such as analytics. The project implementation can be divided into multiple steps as shown in the below slides.

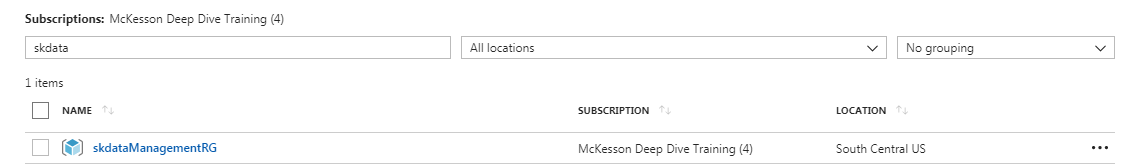
* Create resource group which holds all the remaining components required for the applicatio
* Create a sql server and a sql database
* Create tables and multiple users with access to the newly created database
* Create azure storage account and container to hold your source data flat files
* Create Azure data factory to load the data from flat file into the SQL database
* Mask the data on the sql server for the required users
* Data viewed by user on Tableau

**Resource Group**

Resource groups are nothing but containers that holds related resources for an Azure solution. They provide a way to monitor, control access for the collection of assets that are required to run an application.

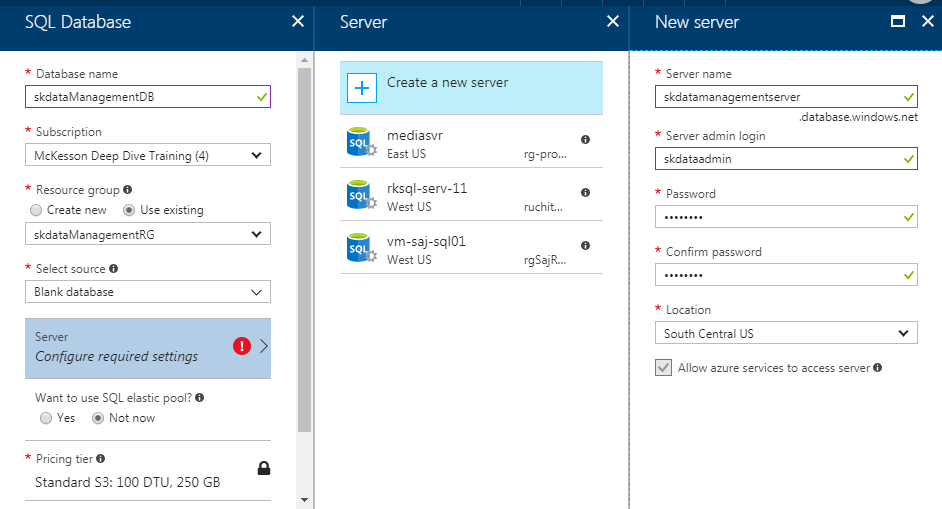
The resource group created was **skdataManagementRG** using **South Central US** as resource group location

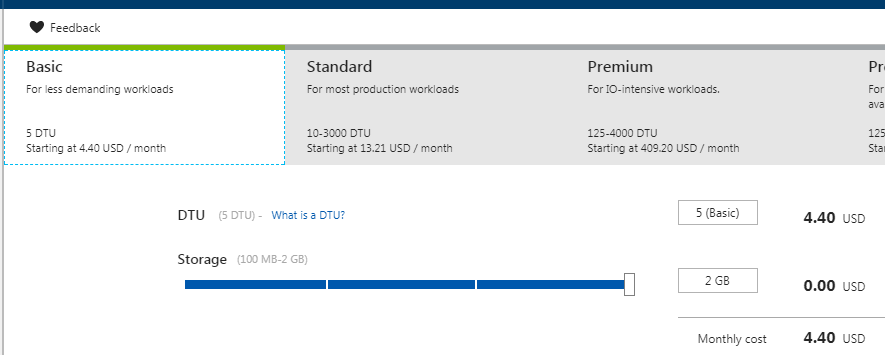




**SQL server database**

* The intention of this process is to showcase the data movement from a flat file to a sql data base that has been hosted on Azure using the technologies provided out of box.
* First and foremost step would be to create a SQL server and SQL database which runs on a cloud computing platform. – change this to remove first and foremost
* We will be creating the database as a blank database so as to have the option of adding our own tables – change this as well
* The server can be created and configured as a part of the creation of the database
* Select the most basic pricing tier considering this is just an one time effort of moving the data and a whole slew of transactions would not be involved



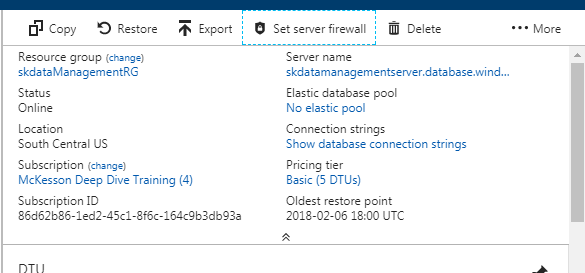


**Firewall setup**

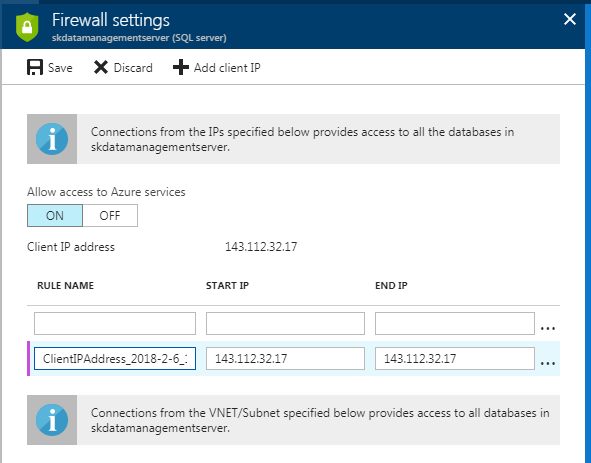
Firewalls are setup to prevent access to the database server except for the client that needs access. To grant access to our client, we setup the firewall on the server

Initially, all Transact-SQL access to your Azure SQL server is blocked by the firewall. To begin using the sql server that we just created we need to specify one or more server-level firewall rules that enable access to our Azure SQL server. Use the firewall rules to specify which IP address ranges from the Internet are allowed, and whether Azure applications can attempt to connect to your Azure SQL server.

* To set a server level firewall rule from the database page, click on “Set Server Firewall”



* Click on “Add Client IP” to add the IP address of the computer you are using currently and click “Save”



**Database users and tables**

The creation of tables and users can also be done using the query editor on the portal directly. In this context, we would be using the management studio client on the local machine to achieve the same.

This project uses 3 tables

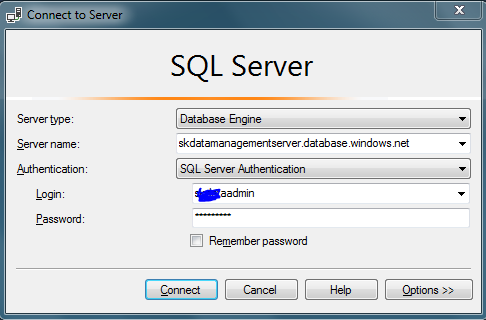
**Patient** – Contains Patient information (Name, DOB, RACE, SSN, Address, Phone, Email)

**Provider** – Contains doctor information (Name, DEA, NPI, Primary and Secondary license numbers)

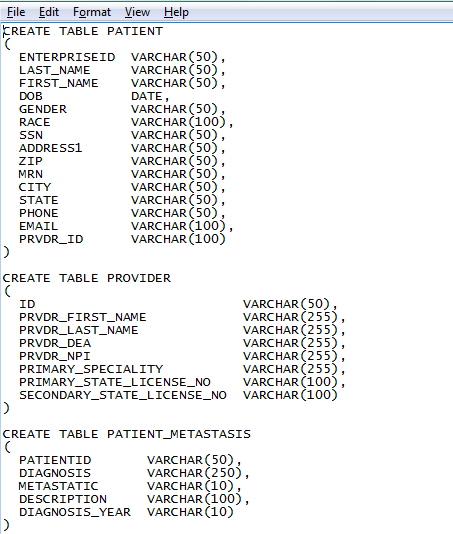
**Patient\_Metastasis** – Contains the patient diagnosis information with a referential integrity to Patient table

* Connect to the SQL database using the local SQL server management studio 2017.
* Once connected using the admin user name and password, crate the required tables on the database
* Create 2 users to demonstrate the data view when masked using dynamic data masking in Azure
* Please remember to provide the fully qualified path of the server to connect using SSMS

For eg skdatamanagementserver.database.windows.net would be the fully qualified name



* **DDLs used in the project**

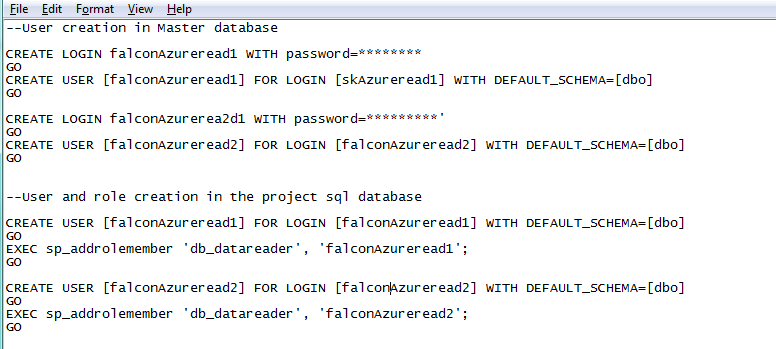


* **User creation script**

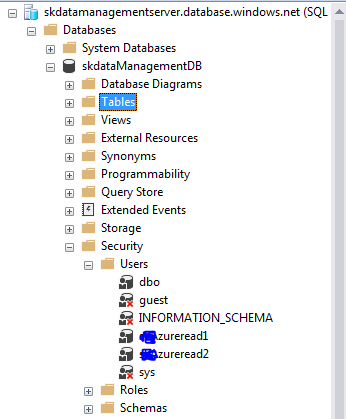
In the context of this project we are creating 2 users. One who will be an authorized user with view of the base data and other as a restricted user

**User** **falconAzureRead1** – with access to view the base data as is, this user will be excluded from masking view and he/she will have the same data view access like the admin in our case

**User** **falconAzureRead1** – with access to view only the masked data



* Once the script runs, the users have been added successfully in the database

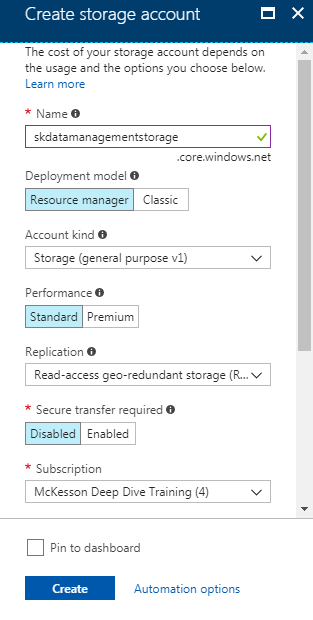


**Azure storage account and container**

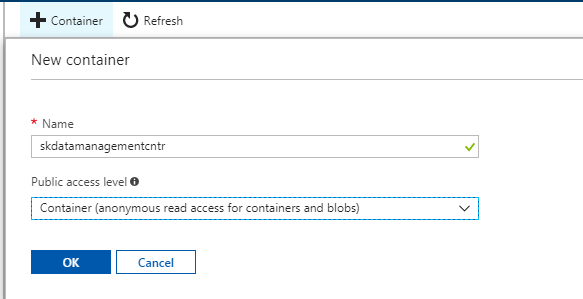
A general purpose **storage account** in azure gives us access to azure storage services such as tables , queues, files, blobs and azure virtual machine disks under a single account. Under this Azure storage account,

In the storage account we create a **storage container** , which holds our blobs. This is where we are going to upload and store the files that contains our project data. Essentially blob storages are used for storing large amounts of unstructured object data. Since the data that we are using is in the range of 4-6 MB, we will be using blob storage.

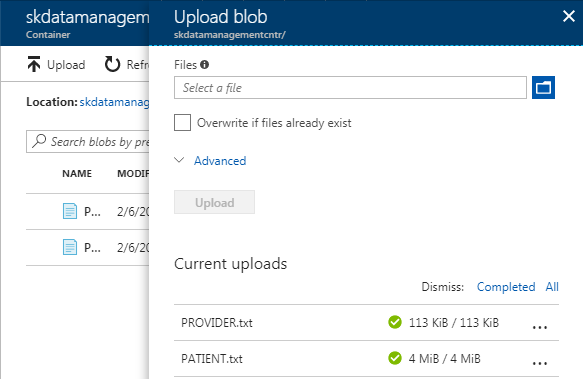
* Creating the storage account



* Creating the storage container to hold our file



* Uploading the data files to the container

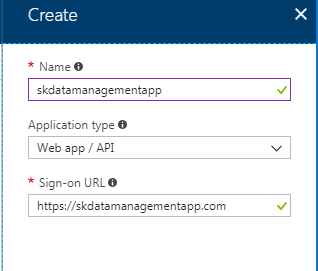


**Application creation**

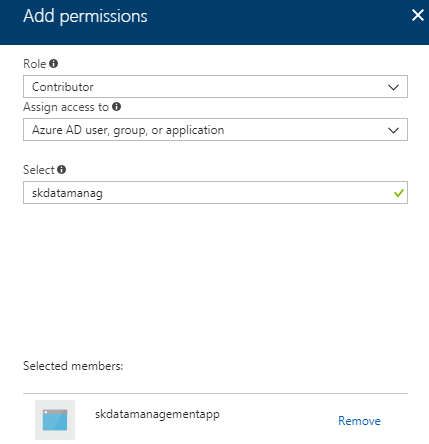
In the context of this project we will be developing a visual studio dotnet application (C# solution) to access the files uploaded in the container and to move them to the tables in SQL server database. We will also be creating the azure data factory through the application. To integrate the application with Azure AD, we must first register the application with Azure AD.

Follow the below steps to create and register an application

* Create the application in azure active directory
* Assign the role of contributor to your subscription to the newly created app
* Get the keyvalue, directory id and application id which needs to be used in creation of the data factory in the future steps
* Creating the application



* Granting access/permission to the created application



**Azure Data factory V2**

.NET Solution for data movement using Azure Data factory v2

In the previous step, we created an Application and registered with Azure AD. We will be using the key values ad IDs from that in our .Net solution. The main purpose of this .Net solution is to move the data from the azure blob storage to the tables on sql server.

We will be creating and using **Azure data factory** V2 for this data movement. **Azure data factory** is a cloud based data integration service which allows us to create data driven workflows in the cloud for automating data movement and data transformation.

The steps to create data factory have been obtained from this Microsoft link

<https://docs.microsoft.com/en-us/azure/data-factory/quickstart-create-data-factory-dot-net>

Before delving into creating the .Net application, keep the below things saved somewhere.

* Keep the directory id, application (tenant id), secure key from the application in the previous step handy
* Pull the access key for the storage account

The .Net solution achieves the data movement by creating and executing the following components

* Creating the data factory – See above paragraph for more information
* Create a linked service – Linked services are created in a data factory to link the data stores and compute services to the data factory
* Create a data set – The data set represents the data to copy from a source.
* Create a pipeline
* Create a pipeline run
* Monitor the pipeline run

The code for doing all these activities is pasted below

To begin with

* **Create a Visual Studio project**
  + Launch Visual Studio.
  + Click File, point to New, and click Project.
  + 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.
  + Enter ADFv2QuickStart for the Name.
  + Click OK to create the project.
* **Import 3 NuGet packages to invoke sql client for the data to be loaded and for data movement**
  + Click Tools -> NuGet Package Manager -> Package Manager Console.
  + In the Package Manager Console, run the following commands to install packages:

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

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

**Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory**

**Code**

**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 ConsoleApp1**

**{**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**// Set variables**

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

**string applicationId = "4c3a256e-8dd3-4ae8-bc92-14d204c72415";**

**string authenticationKey = "qF0gE4stnGwWy83ahcb1DsVeLpHPN9bXutpKYZmTrTk=";**

**string subscriptionId = "86d62b86-1ed2-45c1-8f6c-164c9b3db93a";**

**string resourceGroup = "skdataManagementRG";**

**string region = "East US";**

**string dataFactoryName = "sk06dataFactory";**

**// Specify the source Azure Blob information**

**string storageAccount = "skdatamanagementstorage";**

**string storageKey = "ZJ0G+KcTyfRyqHKLrIUrcILr3DzT2olIX5fT9RvUfg71svroLWkvMDDjsRrYUm+tqf4+dwdZxLcEJntF2tHCiQ==";**

**string inputBlobPath = "skdatamanagementcntr/";**

**string inputBlobName = "PATIENT\_METASTASIS.txt";**

**// Specify the sink Azure SQL Database information**

**string azureSqlConnString = "Server=tcp:skdatamanagementserver.database.windows.net,1433;Database=skdataManagementDB;User ID=skdataadmin@skdatamanagementserver.database.windows.net;Password=admin@12;Trusted\_Connection=False;Encrypt=True;Connection Timeout=30";**

**string azureSqlTableName = "dbo.PATIENT\_METASTASIS";**

**string storageLinkedServiceName = "AzureStorageLinkedService";**

**string sqlDbLinkedServiceName = "AzureSqlDbLinkedService";**

**string blobDatasetName = "BlobDataset";**

**string sqlDatasetName = "SqlDataset";**

**string pipelineName = "skTutorialBlobToSqlCopy";**

**// 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 = "RACE",**

**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 = "PATIENTID",**

**Type = "String"**

**},**

**new DatasetDataElement**

**{**

**Name = "DIAGNOSIS",**

**Type = "String"**

**},**

**new DatasetDataElement**

**{**

**Name = "METASTATIC",**

**Type = "String"**

**},**

**new DatasetDataElement**

**{**

**Name = "DIAG\_DESCRIPTION",**

**Type = "String"**

**},**

**new DatasetDataElement**

**{**

**Name = "DIAGNOSIS\_YEAR",**

**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...");**

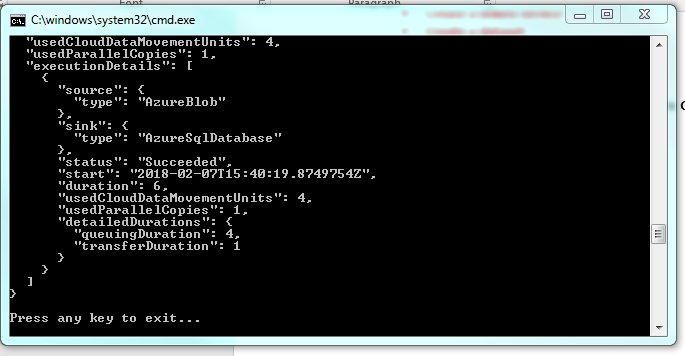
**Console.ReadKey();**

**}**

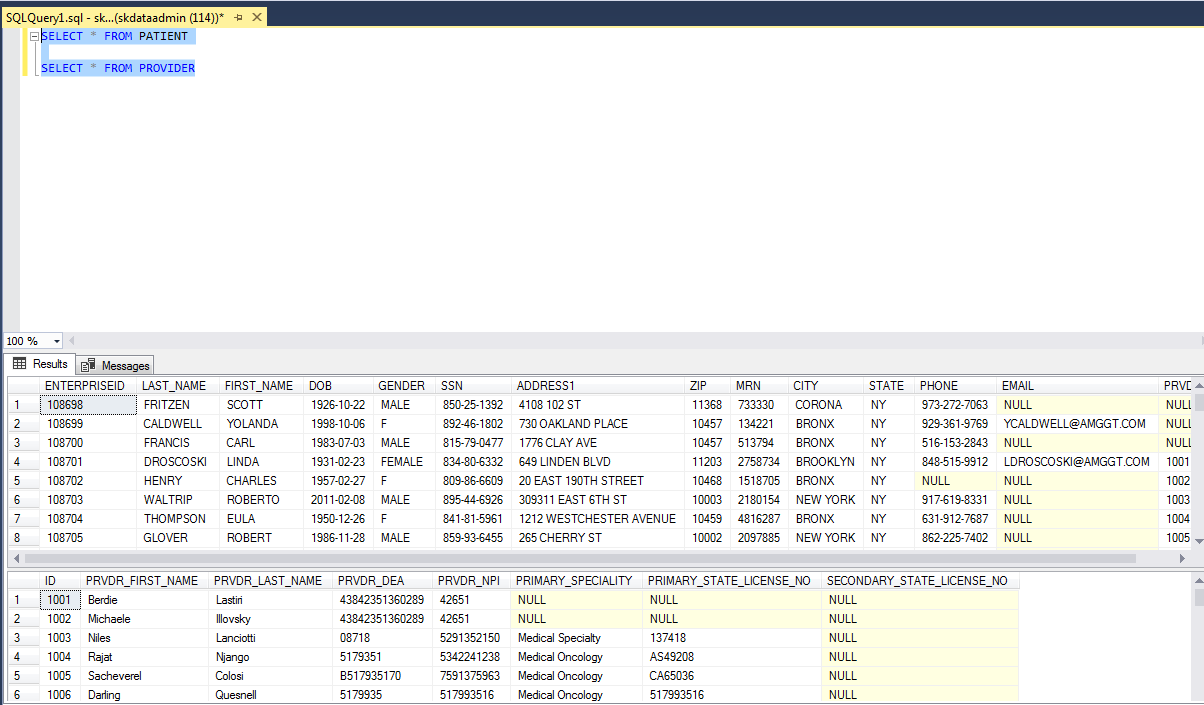
**}**

**}**

* **Run the code and check the output**



* **End of the run data will be loaded into the tables on sql database**



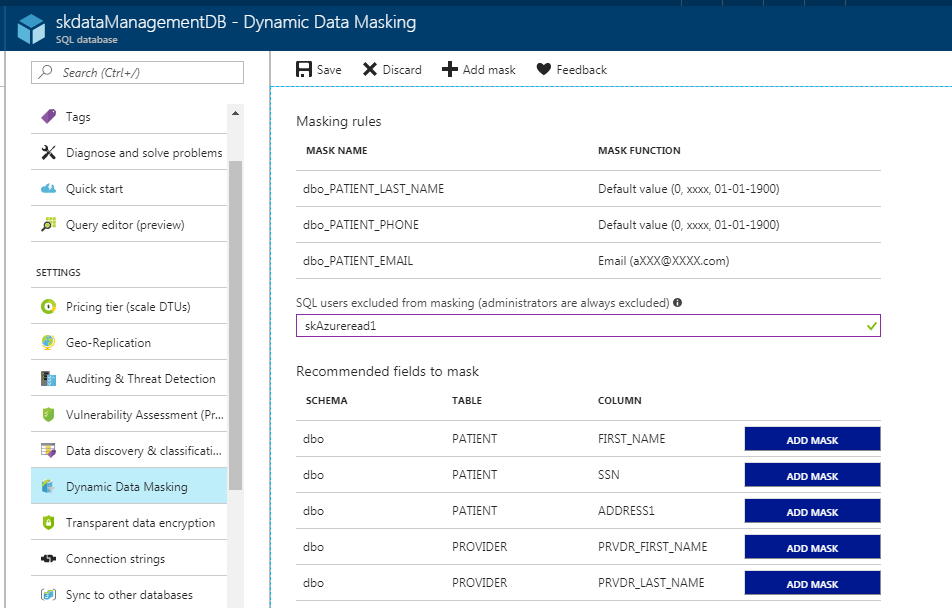
**Data masking**

This is where we look at the data in our tables on the portal and decide to mask the PHI/PCI/PII columns as well any sensitive information. For our purpose, we are masking the last name, date of birth, social security number, email, phone number, medical record number for the patient. We are also masking the provider last names.

For the purpose of demonstration, we will be excluding the user **falconAzureread1** from masking which essentially means, this user will have the full true view of the base data. The other user who is not excluded will only have the masked data view

Follow the steps below to mask the tale data

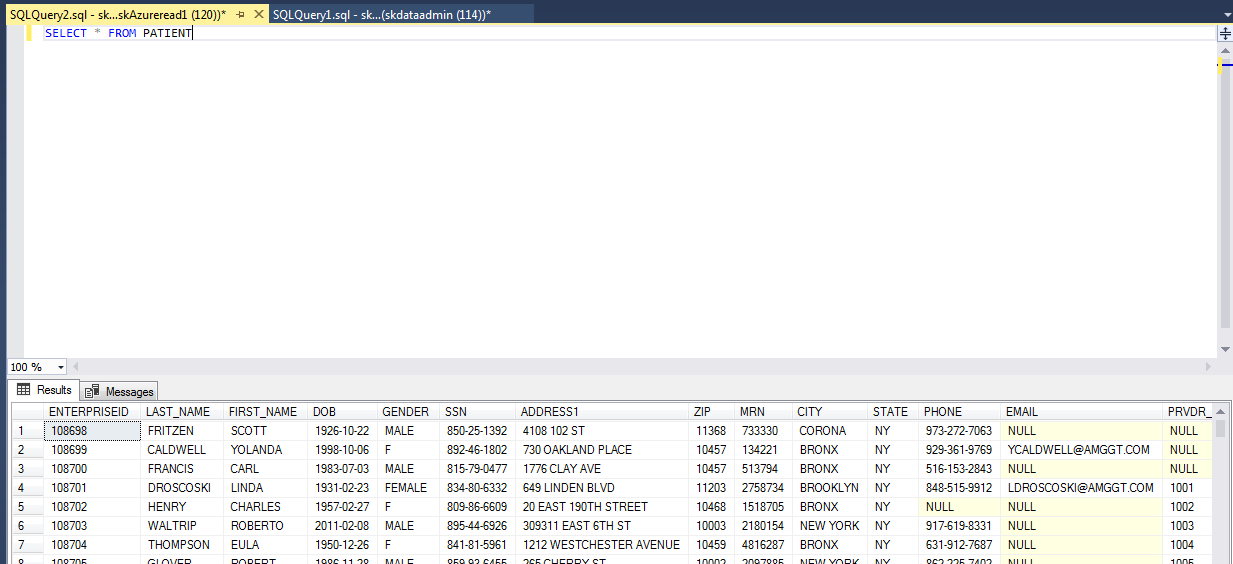
* Look up the tables that was loaded using the dot net application in azure portal
* Use the dynamic data masking option provided out of box by Azure to change the view of data for few users
* You have the option of excluding the a user from the list of users who would still have full view of the actual data
* Login as different users to check the masking



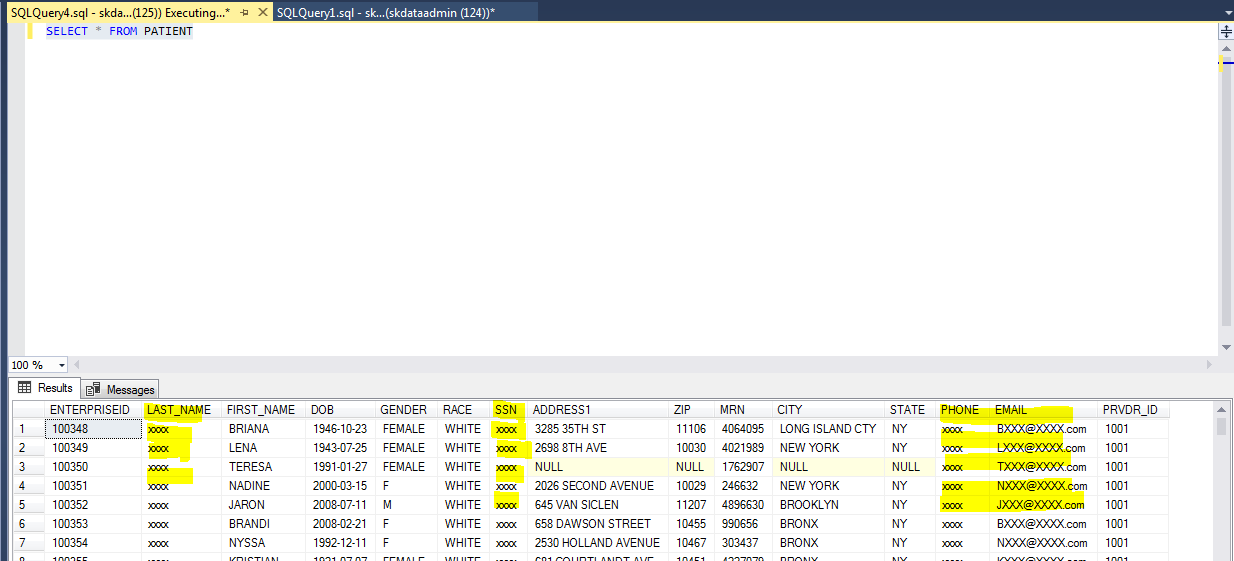
**Final Results**

To show case our final results of masking, we are going to see the different views of data when logged in as different users

* Logging in as **falconAzureread1** user who has been excluded from masking. The admin account is by default excluded from masking
* **falconAzureread1** will have the actual view of the data and not the masked values



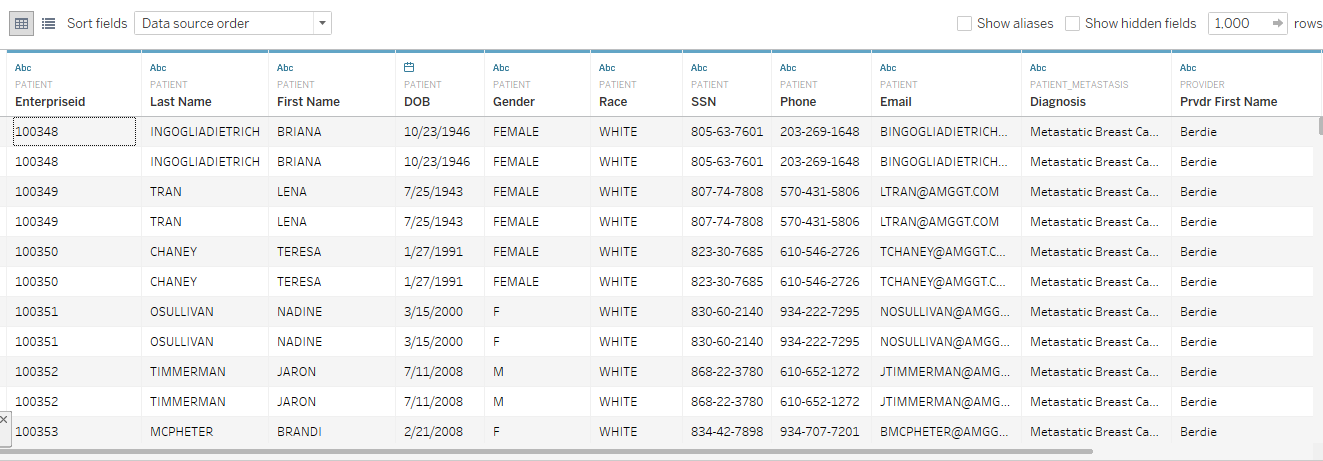
* Logging in as skAzureread2 user who has been included to view only the masked data. The admin account is by default excluded from masking
* skAzureread2 will only see the masked data on the columns where masking was applied.



**Data Visualization**

Tableau is our tool of choice to show the data visualization. The main purpose for this is to just show and prove that data masking will not affect any down stream capabilities on data like analytics. In this project, an attempt is made to show the cancer diagnosis by race and by year as well. The data for this was mocked up based on the statistics provided by CDC.

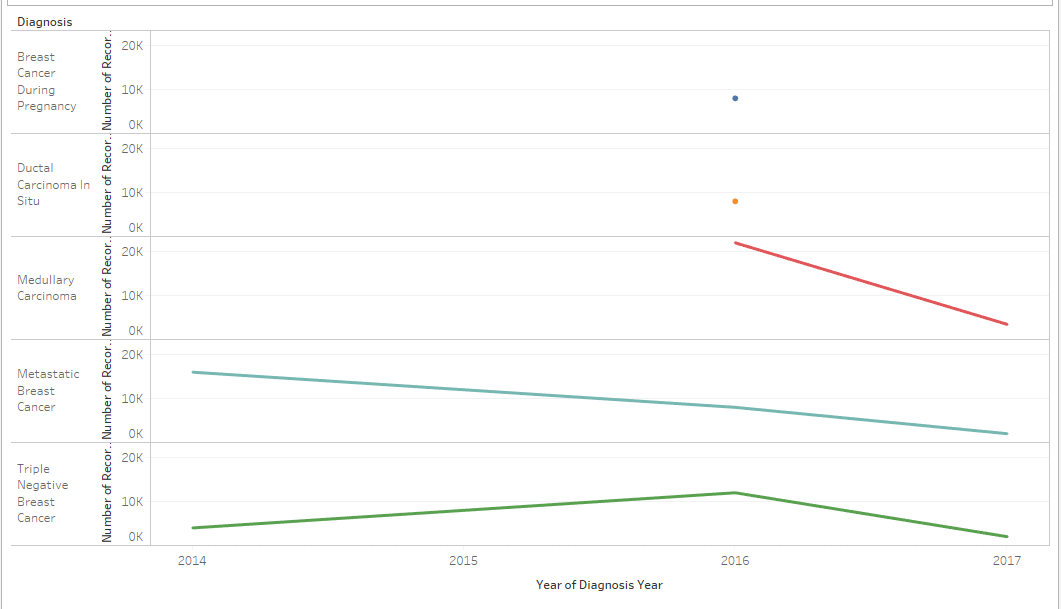
The first visualization for data is based on skAzureread1 user who was excluded from masking. The user sees the base data as is.



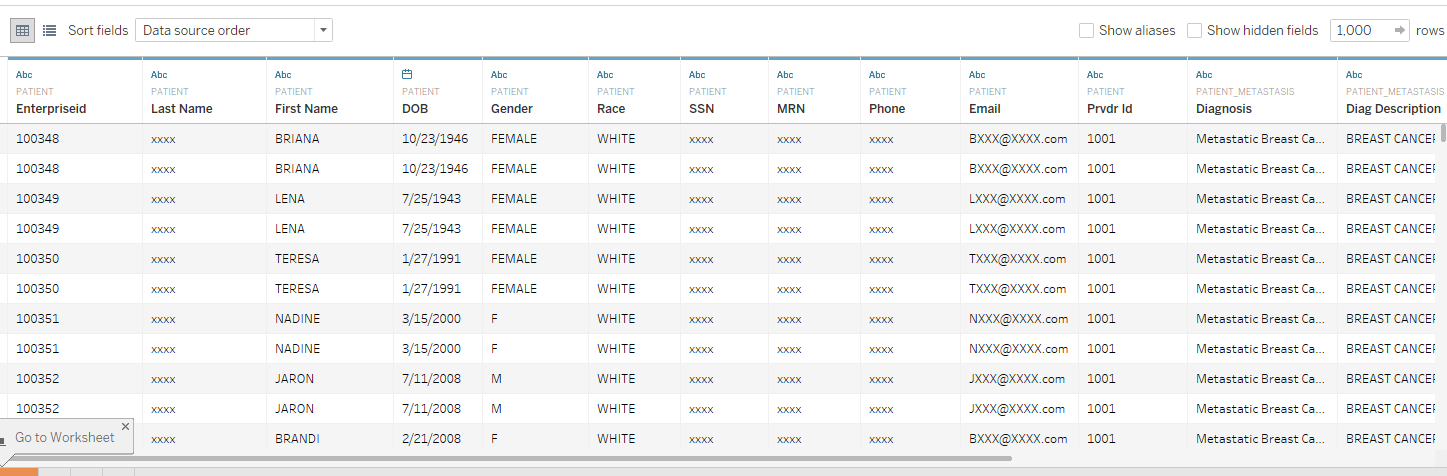
Graph shows number of different cancer diagnosis by year and type of cancer



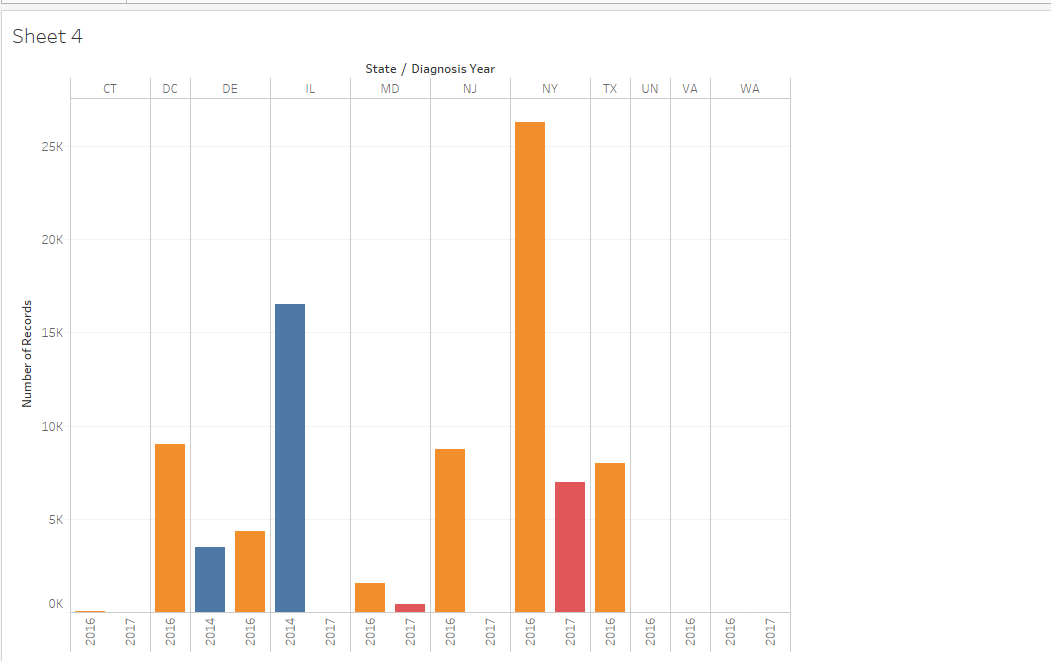
Same data in a different format



The second visualization for data is based on skAzureread2 user who only has access to the masked view of data. He sees the masked data in Last name, SSN, MRN, PHONE and email columns (XXX instead of the real values)



Even without being able to see the actual patient data, the user is still able to run a report on the number of cancer diagnosis per state and year of diagnosis



**Lessons Learned**

Data masking does not hurt the downstream activities on the data. 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 and simplicity of Microsoft GUI
* 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

**Cons**

* Data cannot be masked using custom dictionaries
* Limited customization available out of the box for dynamic data masking
* Only data view changes not the actual underlying data. It will be very apparent that the data has been masked

**Future Plans**

* Need to research more how to mask the data when it is present in a sql data warehouse
* Options for creating a data lake and masking that data needs to be researched as well

**URLs and LINKS**

* Github link -- <https://github.com/sujathakestur/Data-Management-and-Masking>
* Youtube 2 minute video -- <https://youtu.be/HMTSiAkJdRo>
* Youtube 15 minute -- <https://youtu.be/ZOaRrkYNng0> (part 1)

<https://youtu.be/g72kE7V34pw> (part 2)