**Data Management and Masking on Azure**

**Dilip Reddy**

**Problem Statement**:

Cyber security is a big challenge in the current world. Most of the companies has a need to mask enterprise data to protect from outside intrusions and accidental loss of protected data. So, Data Masking is an important technology used to maintain data security even in case of external threats. Data exposure to the right user is essential to properly organize and maintain data processes to meet ongoing information lifecycle needs.

**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. You enable DDM by defining masking rules on designated database fields, which determine how you want the data in these fields to appear in query results. You can define a partial mask which exposes some of the data in the selected field such as the first and last few characters, while masking out the rest. Or, you can define a full mask which doesn't leave any of the data exposed and always replaces the field's data with a constant value.

**High Level Steps:**

1. Create a SQL server and a sql database
2. Create tables and multiple users with access to the newly created database
3. Create azure storage account and container to hold your source data flat files
4. Create Azure data factory to load the data from flat file into the SQL database
5. Mask the data using Azure Data Masking service on the sql server for the required users
6. View the data on Tableau or any other visualization tool

**Data set obtained at**:

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

**Hardware/OS**:

Intel Core i5-5300U CPU 2.30 GHZ, 16 GB RAM, 64 bit Windows 7 operating system

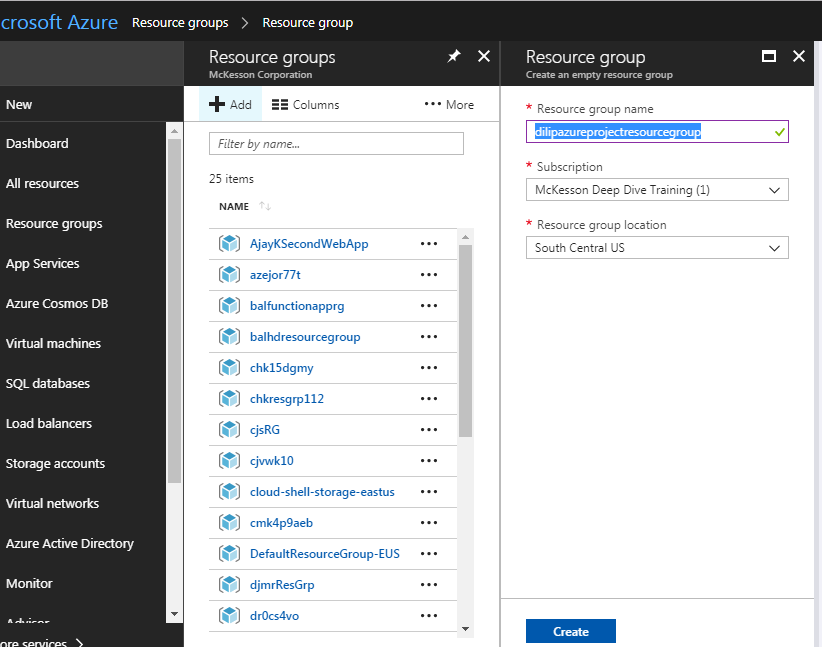
**Software used**:

Visual Studio 2017, C#, SQL Server Management studio 2017, Tableau

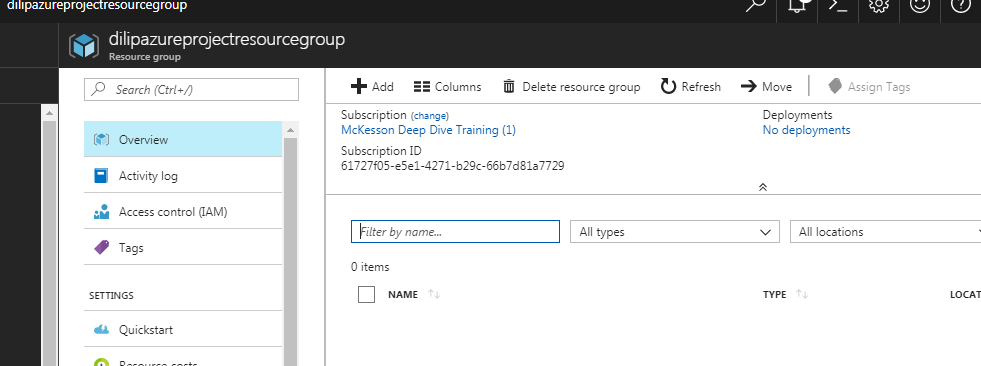
1. **Prerequisites**
2. Access to azure portal
3. Download SQL Server Management Studio 2017 on to your local machine
4. Download visual studio 2017
5. Identify dataset to be masked
6. **Implementation Steps**
   1. **Resource Group Creation**

Resource group is a container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group.

1. Login to Azure portal and create a resource group ‘dilipazureprojectresourcegroup’
2. Enter Resource group name as ‘dilipazureprojectresourcegroup’
3. Subscription as ‘McKesson Deep Dive Training (1)’
4. Resource group location as ‘South Central US’
5. Click on Create



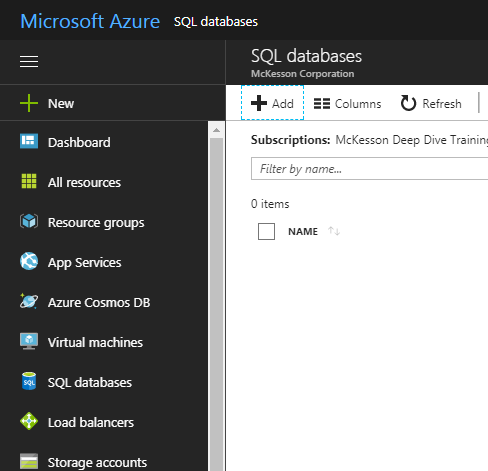
Resource group created



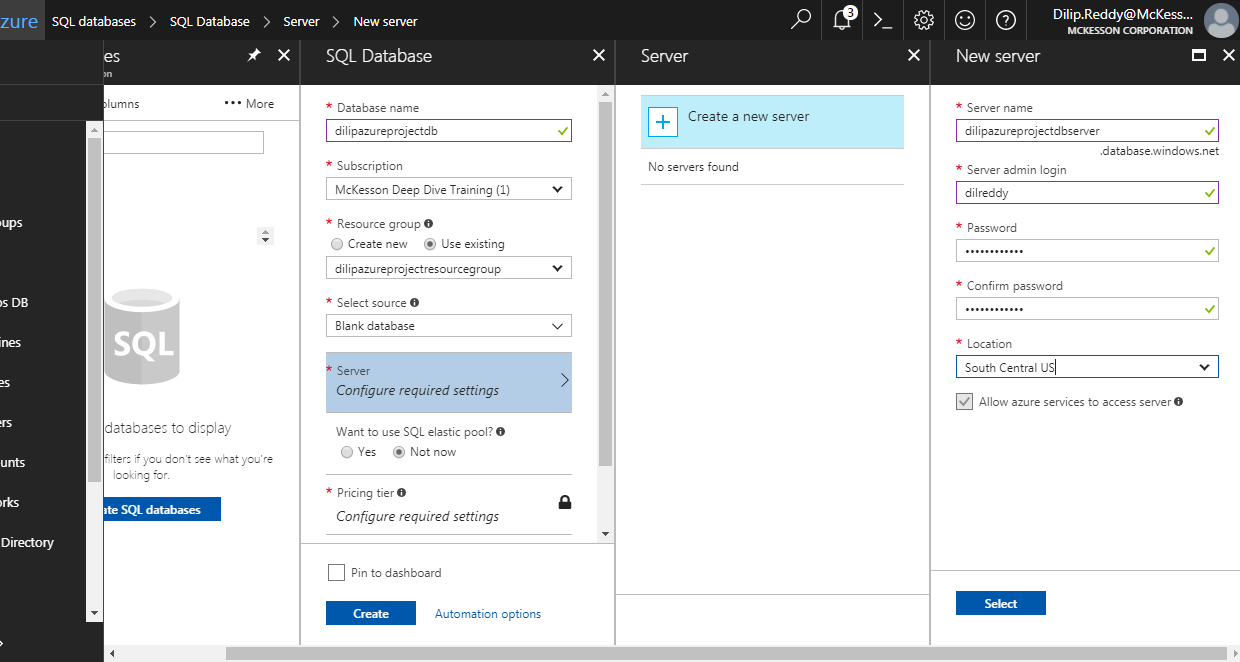
* 1. **SQL DB/ SQL Server Creation**

Azure SQL Database is a relational database-as-a service using the Microsoft SQL Server Engine. SQL Database is a high-performance, reliable, and secure database you can use to build data-driven applications and websites in the programming language of your choice, without needing to manage infrastructure.

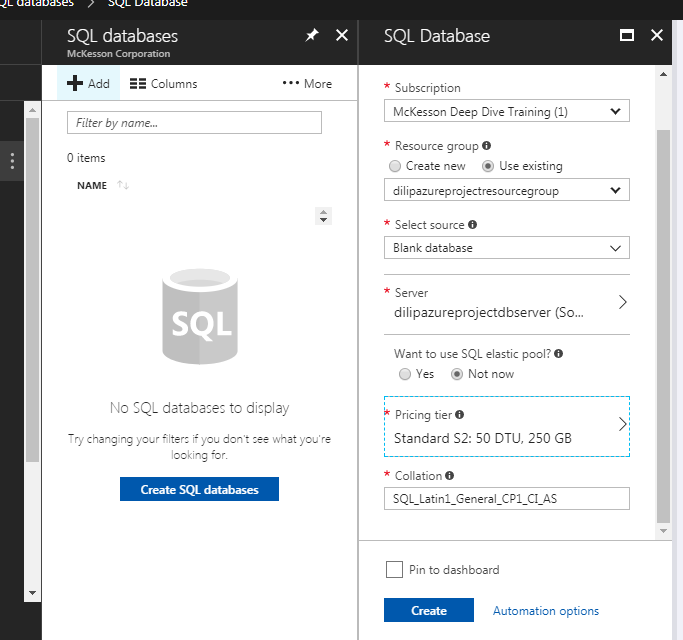
On the left side under services click on SQL databases



1. Click on Add
2. Enter database name
3. Select Subscription, Resource group
4. Select Blank database under Select Source
5. Click on Server to create a new server
6. Click on Create a new server
7. Enter Server name, Server admin login, password and Location
8. Click on Select button

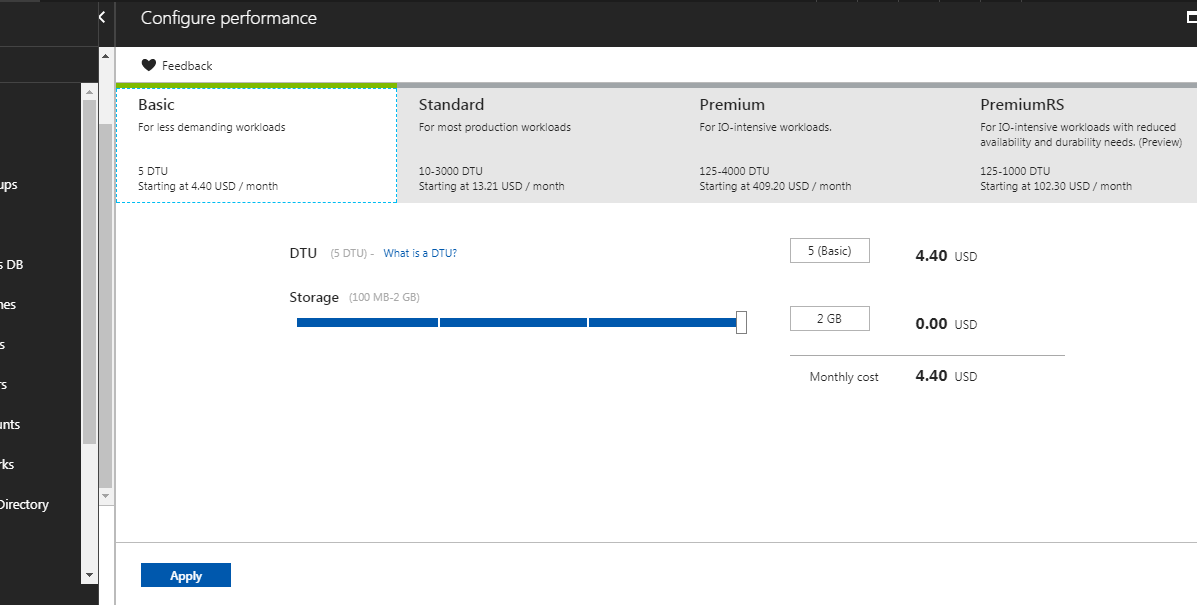


After the server is created it will assign to SQL Database under Server

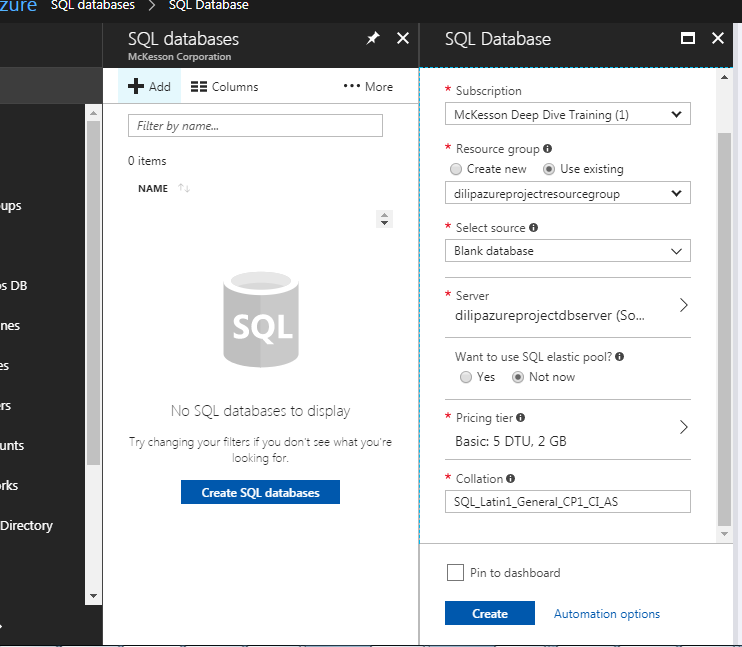


Click on Pricing tier and select Basic and click on Apply

*Note: For this project Basic is enough, but depending on your project you should select a different pricing tier*

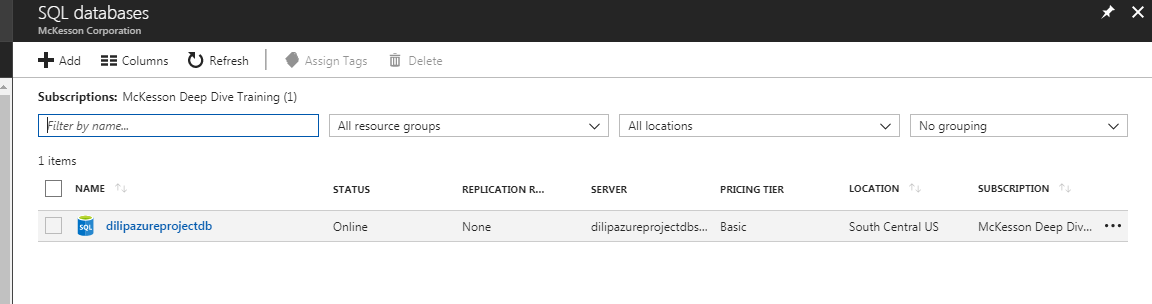


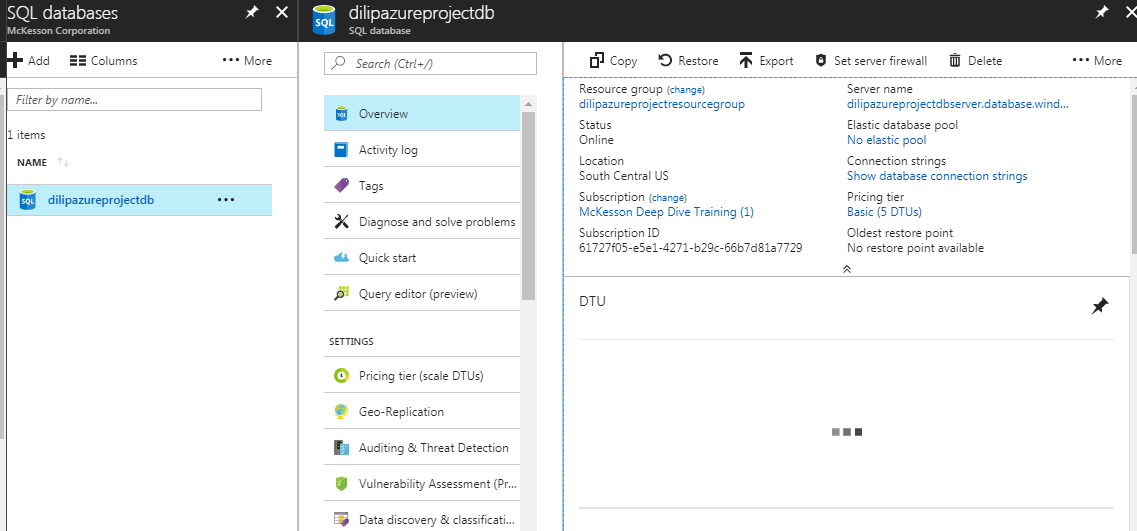
Basic pricing tier is assigned under ‘Pricing tier’ as shown in below screen shot.



Click on Create, it takes a while to create sql db.

dilipazureprojectdb created

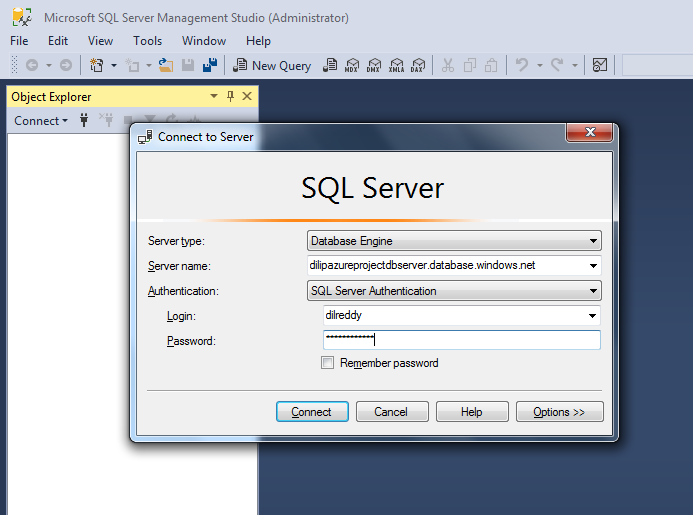




Login to sql server db and make sure you are able to login to the db.

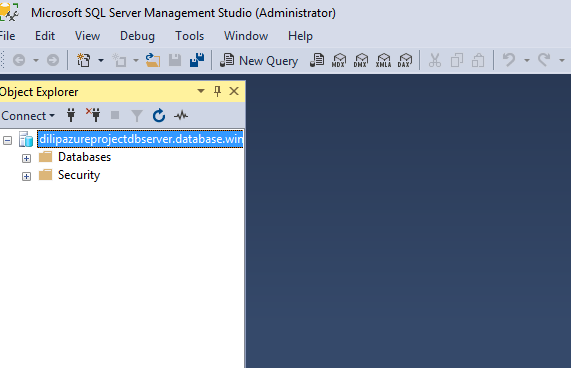
Click on Sql Server Management Studio

Enter the credentials used while creating the db and click on connect



It will ask to create a new fire wall rule. Click on Sign in and then click on ok.

Logged in to the db



After logging in to the sql server db, create Patient and Provider tables.

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 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),

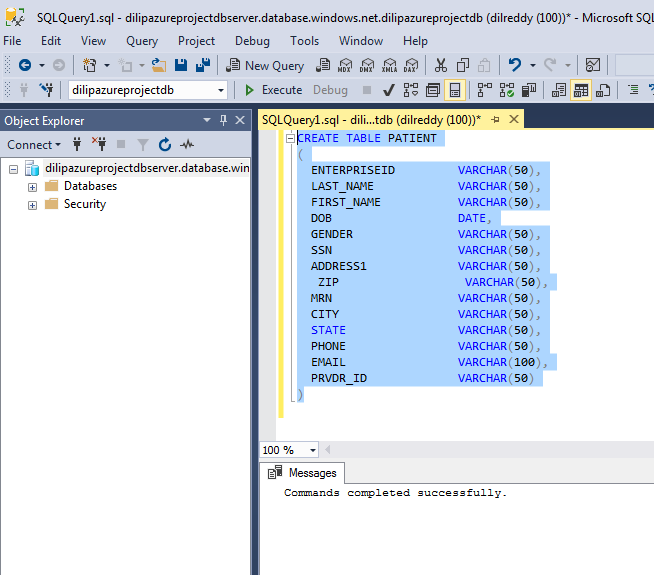
PHONE VARCHAR(50),

EMAIL VARCHAR(100),

PRVDR\_ID VARCHAR(50)

)

Tables created successfully



After creating the tables create two users

User1 is created to include in data masking, that means this user can see only the masked data

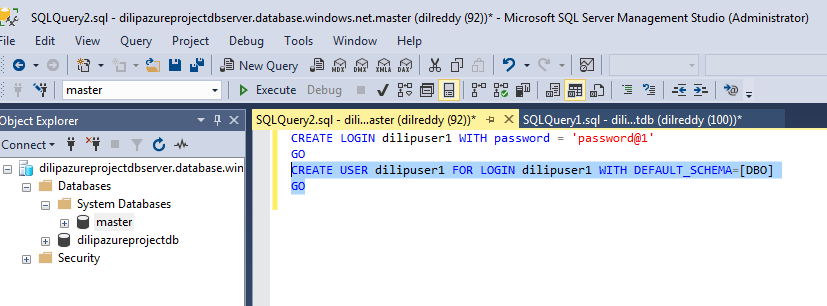
User2 is created to exclude from data masking, that means this user can see all the original data

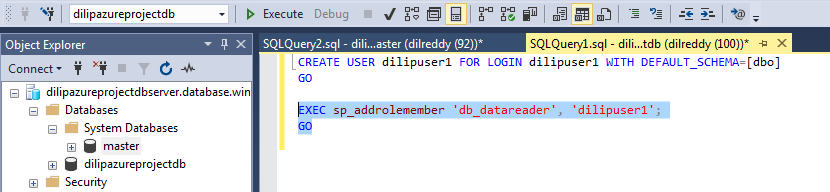
*CREATE LOGIN dilipuser1 with password ‘password@1’*

*GO*

*CREATE USER dilipuser1 FOR LOGIN dilipuser1 WITH DEFAULT\_SCHEMA=[DBO]*

*GO*





Similarly create another user dilipuser2

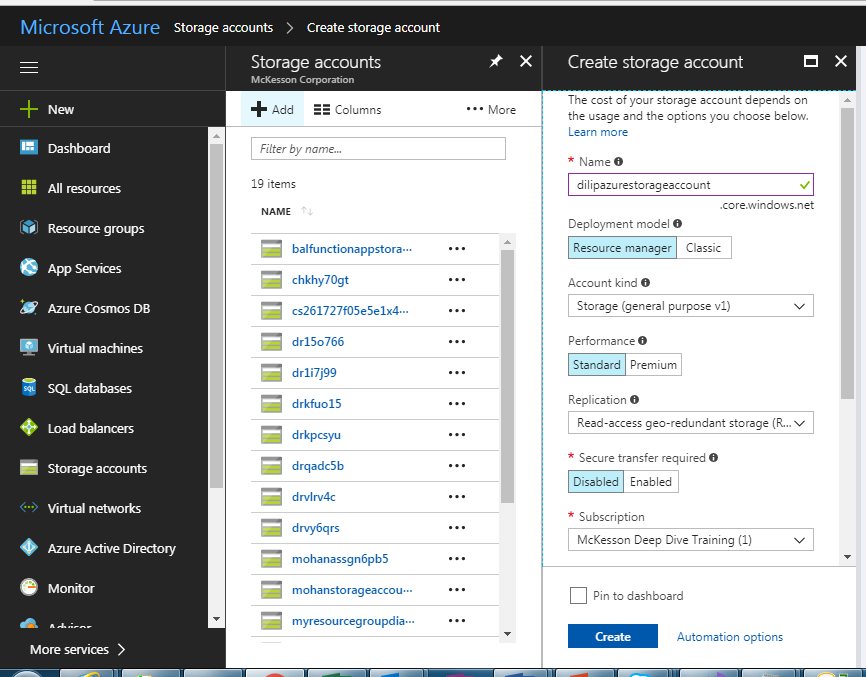
* 1. **Create Storage Account:**

Microsoft Azure Storage is a Microsoft-managed cloud service that provides storage that is highly available, secure, durable, scalable, and redundant. Microsoft takes care of maintenance and handles critical problems for you.

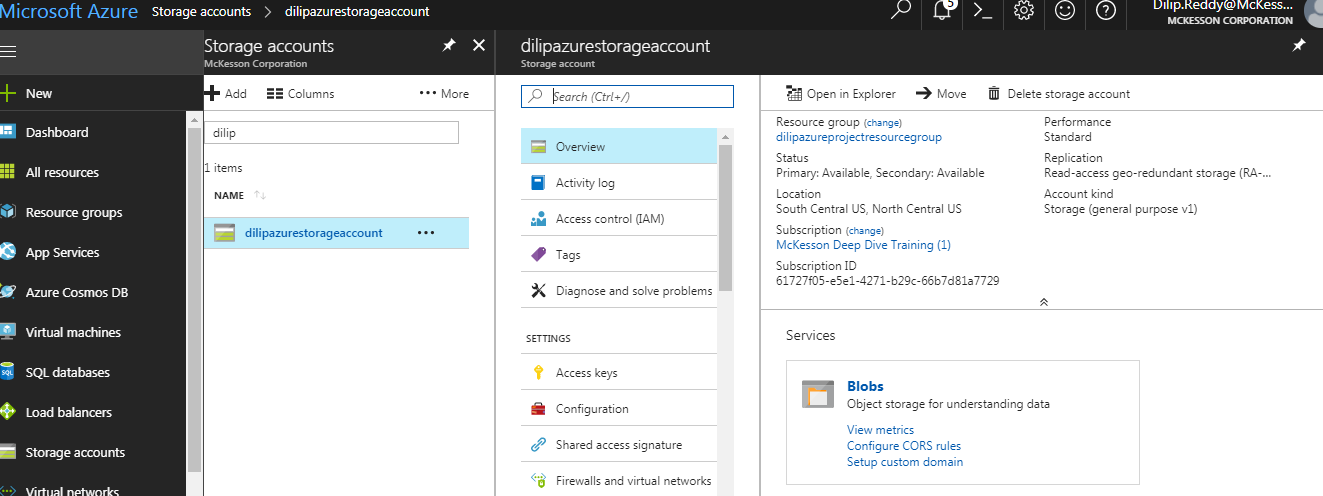
Azure Storage consists of three data services: Blob storage, File storage, and Queue storage.

On the left side under services click on Storage accounts. Click on Add.

Enter all the required information and click on create.

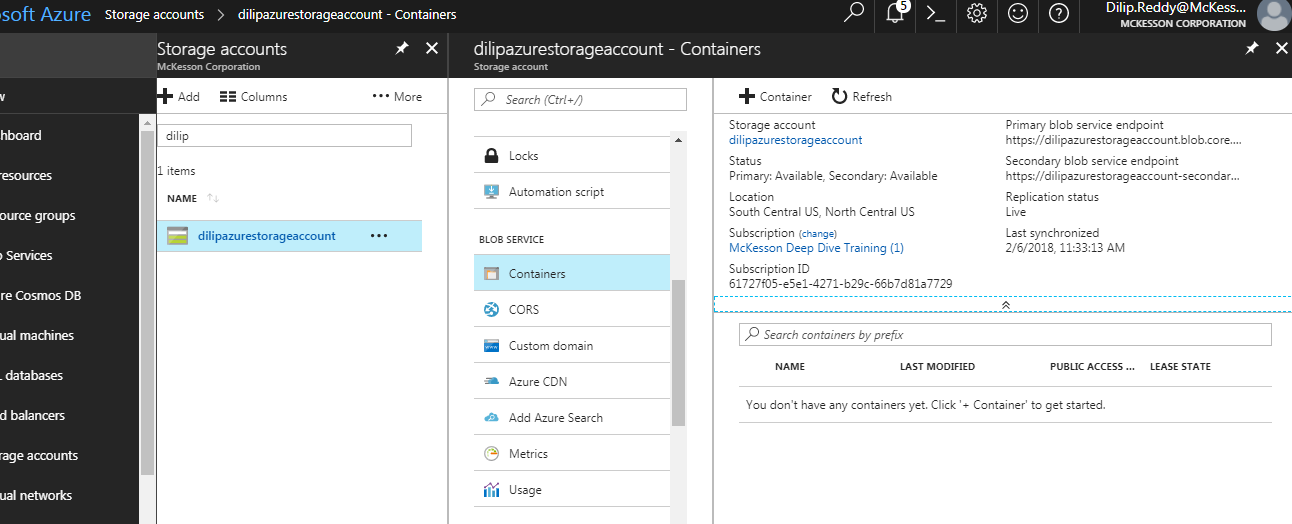


Storage account created

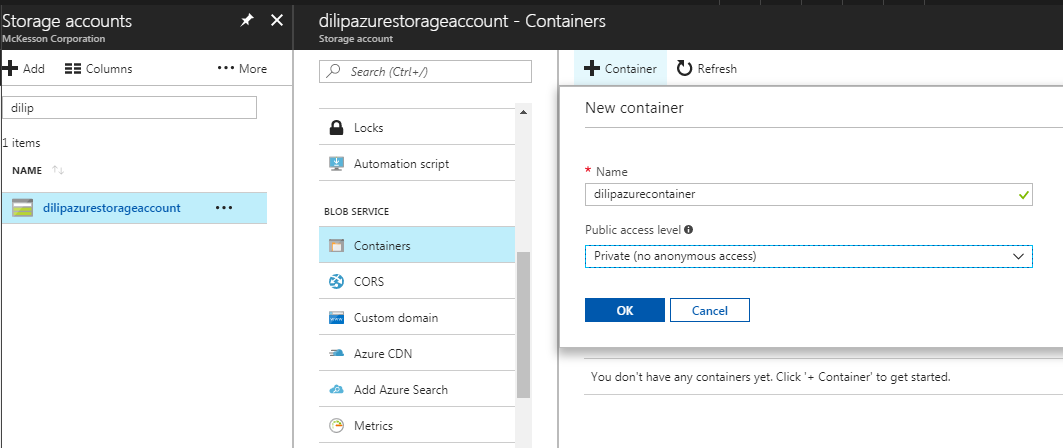


* 1. **Create a Container for the storage account**

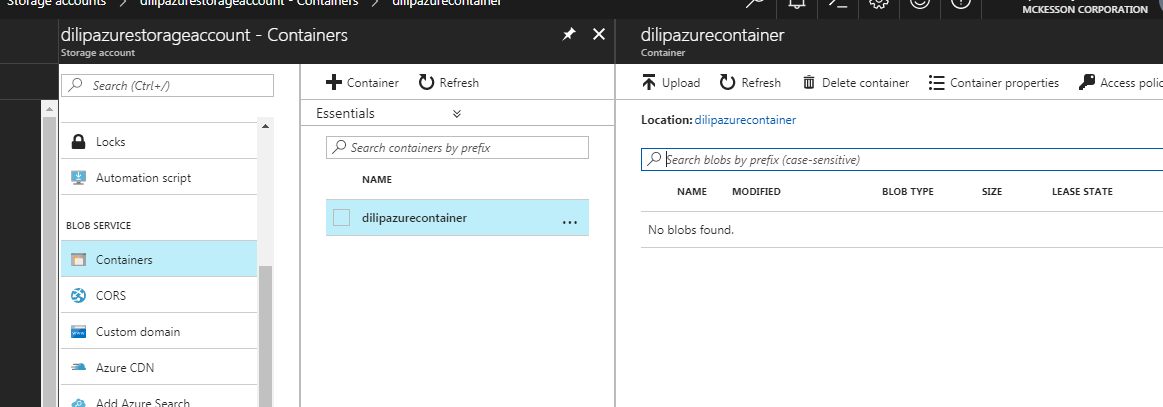
1. Click on storage account created in the above step.
2. Under BLOB SERVICE, click on Containers
3. Click on +Container



1. Click on ok

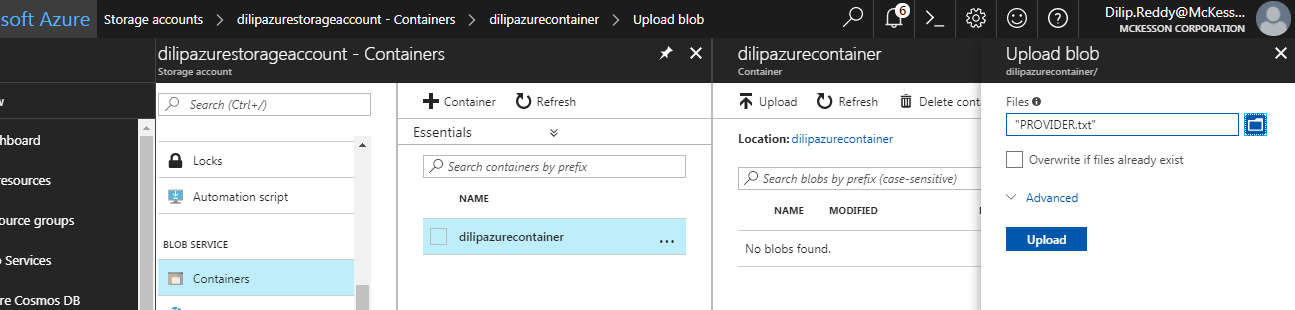


1. dilipazurecontainer created

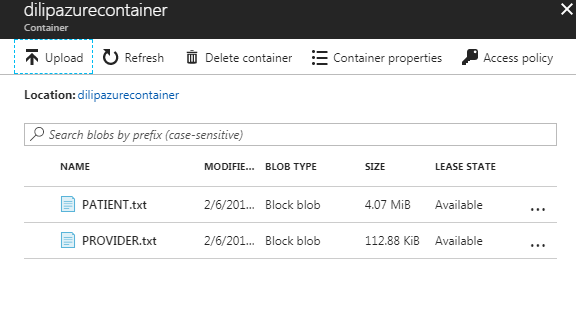


* + 1. **Upload text file to container**

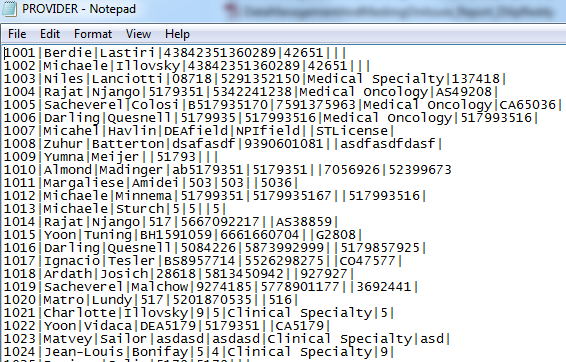
Click on Upload and upload PROVIDER.TXT and PATIENT.TXT file to container

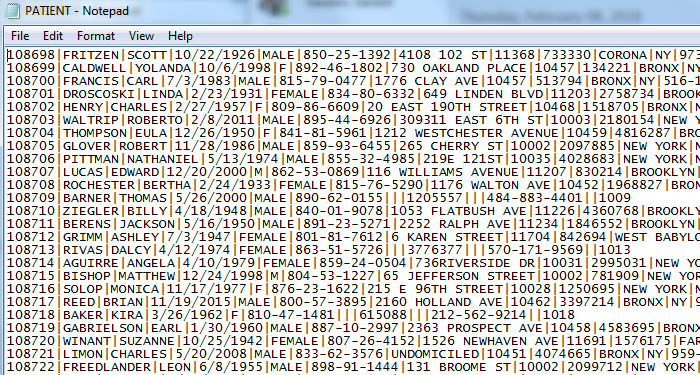


PATIENT and PROVIDER files uploaded to container.



* + 1. **Sample Data Files**

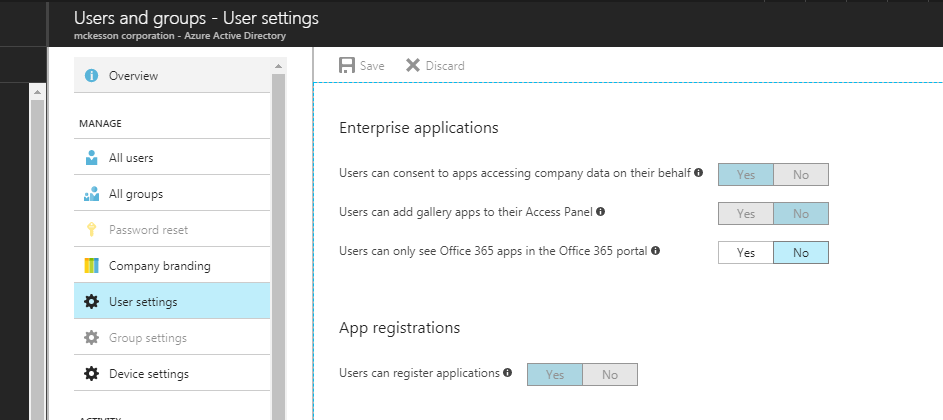


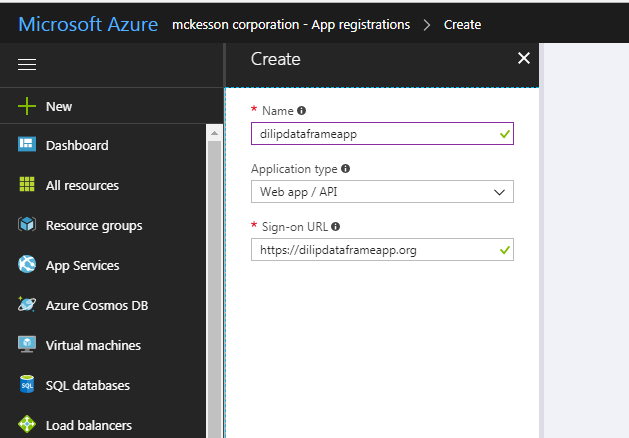


* 1. **App Creation**
     1. **Create an azure active directory application**

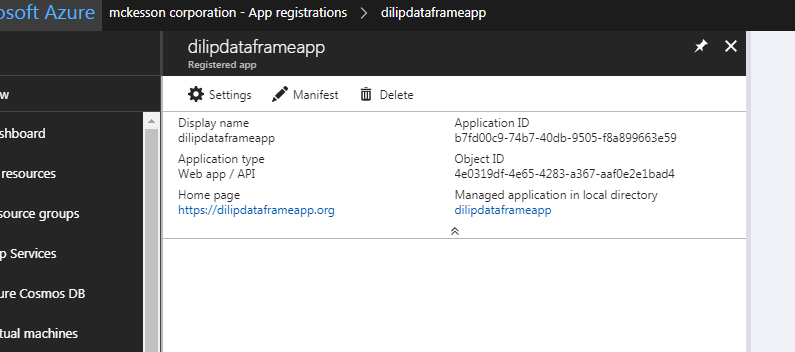
Azure Active Directory (Azure AD) is Microsoft’s multi-tenant, cloud based directory and identity management service. Azure AD combines core directory services, advanced identity governance, and application access management. Azure AD also offers a rich, standards-based platform that enables developers to deliver access control to their applications, based on centralized policy and rules.

1. Go to Active Directory to check if you can register an App
2. Default is set to Yes!
   * 1. **Set up an Azure Active Directory (AD) application and assign permissions**
3. Click on Azure Active Directory under services
4. Click on App Registrations
5. Click on New application registration



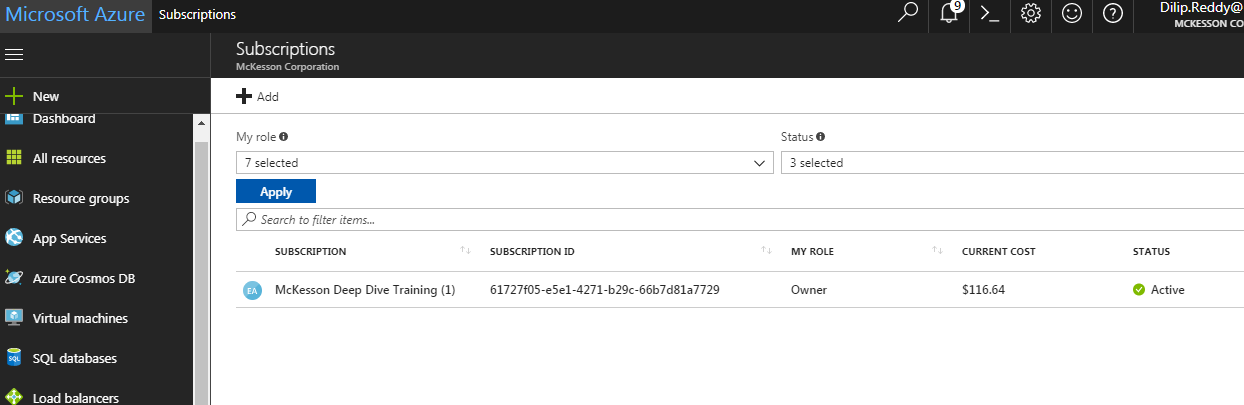


Dilipdataframeapp created successfully

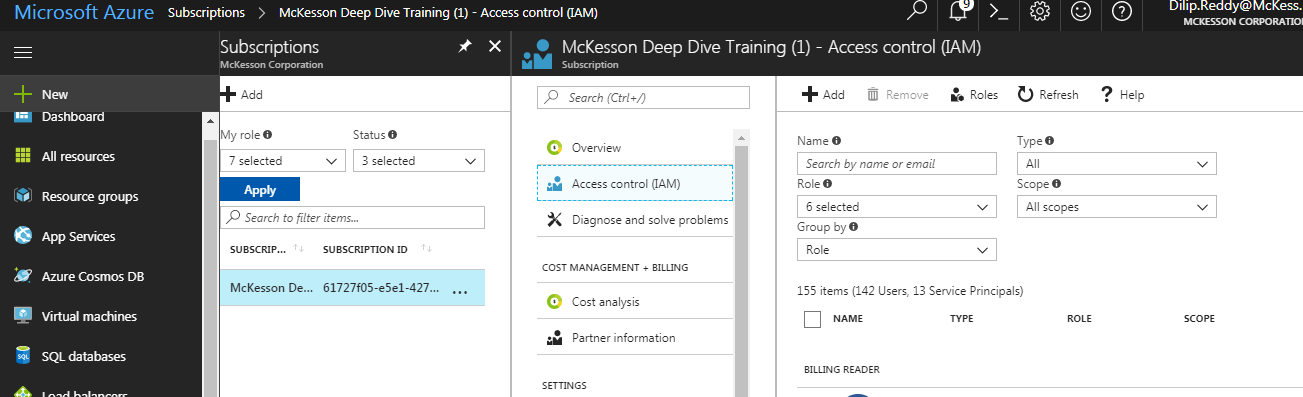


Assign a role to the above app

Under New -> More Services -> Subscriptions



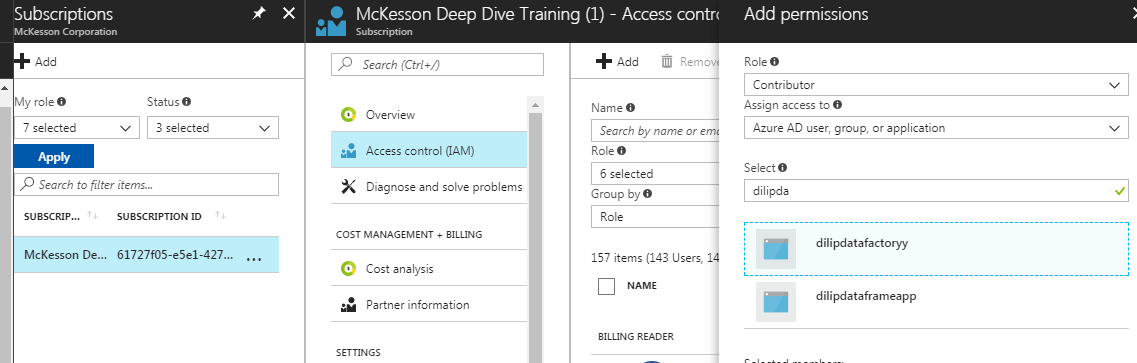
Select your Subscription -> Access Control (IAM) -> Add



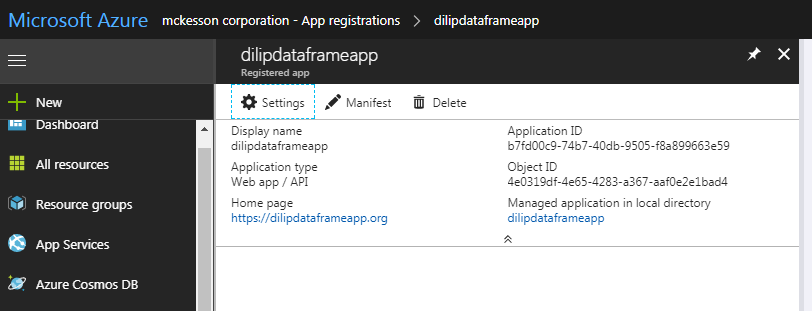
Under Role select Contributor.

Note: You need to have this permission allowed if you are using the McKesson subscription.

Search for your data frame app, select it and then select Save

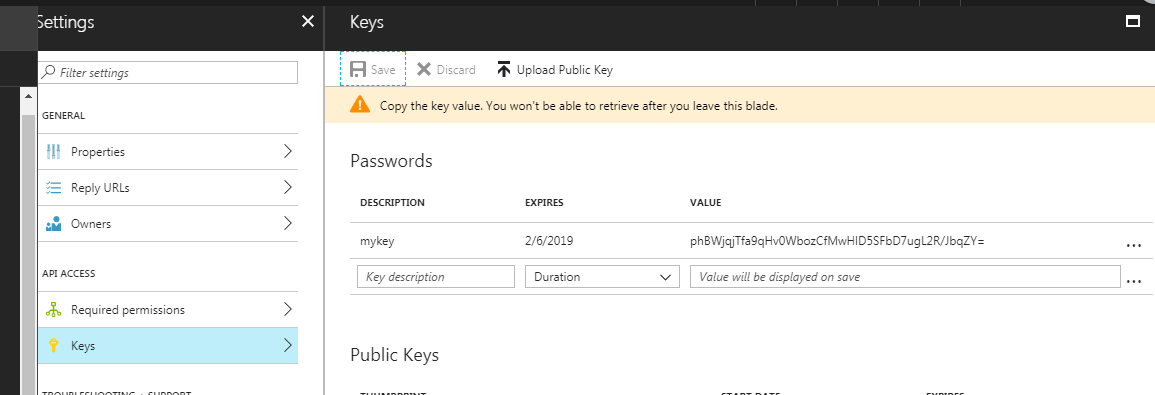


Capture the application id, application id is needed for .NET code.



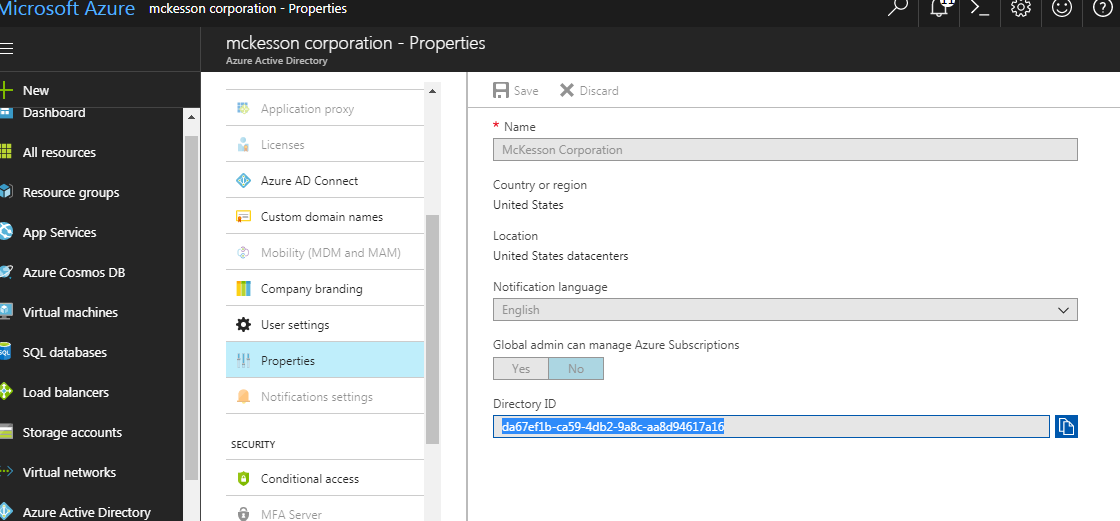
Create authentication key by clicking on settings ->keys

And enter description, expiration and click on save. After clicking on save, the key will display. Save this key, this key will be used in .NET code.



Click on Azure Active Directory->Properties and get tenant id

Tenant ID will be used in .NET code



1. **Data Factory Creation**

The [Azure Data Factory](http://azure.microsoft.com/en-us/documentation/services/data-factory/) (ADF) is a service designed to allow developers to integrate disparate data sources.  It is a platform somewhat like SSIS in the cloud to manage the data you have both on-prem and in the cloud.

Note:*To create Data Factory instances, the user account you use to log in to Azure must be a member of****contributor****or****owner****roles, or an****administrator****of the Azure subscription. In the Azure portal, click your****user name****at the top-right corner, and select****Permissions****to view the permissions you have in the subscription.*

**3.1** **Prerequisites to Create a Data Factory**

1. Install Visual Studio

2. Portal: Obtain your Subscription ID

3. Create an Azure Storage Account & Blob container

4. Portal: Create a data file and upload data file to Blob container

5. Portal: Create an app in Active Directory (App ID, Client Secret Key, Directory ID/Tenant ID).

* 1. **Create a Visual Studio project**

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

Launch **Visual Studio**.

1. Click **File**, point to **New**, and click **Project**.
2. 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.
3. Enter **dilipdfapp** for the Name.
4. 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.

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;

*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

namespace DataFactory

{

class Program

{

static void Main(string[] args)

{

// Set variables

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

string applicationId = "b7fd00c9-74b7-40db-9505-f8a899663e59";

string authenticationKey = "phBWjqjTfa9qHv0WbozCfMwHlD5SFbD7ugL2R/JbqZY=";

string subscriptionId = "61727f05-e5e1-4271-b29c-66b7d81a7729";

string resourceGroup = "dilipazureprojectresourcegroup";

string region = "EAST US";

string dataFactoryName = "dilipdatafactoryy";

// Specify the source Azure Blob information

string storageAccount = "dilipazurestorageaccount";

string storageKey = "usk5E1K7Sj2HfX2V/eZRq/hU5k/oZs0n6WgcixaYK+Im6IKGk0er7vFLTbv6yBe5rNdcEqqpHh1YvNUY3vugvw==";

string inputBlobPath = "dilipazurecontainer/";

string inputBlobName = "PATIENT.txt";

// Specify the sink Azure SQL Database information

string azureSqlConnString = "Server=tcp:dilipazureprojectdbserver.database.windows.net,1433;Initial Catalog=dilipazureprojectdb;Persist Security Info=False;User ID=dilreddy@dilipazureprojectdbserver.database.windows.net;Password=password143#;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30";

string azureSqlTableName = "dbo.Patient";

string storageLinkedServiceName = "AzureStorageLinkedService";

string sqlDbLinkedServiceName = "AzureSqlDbLinkedService";

string blobDatasetName = "BlobDataset";

string sqlDatasetName = "SqlDataset";

string pipelineName = "Adfv2TutorialBlobToSqlCopyy";

*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 };

// 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);

}

*Add the following code to the****Main****method that creates an****Azure Storage linked service****.*

*You create linked services in a data factory to link your data stores and compute services to the data factory. In this Quickstart, you only need to create one Azure Storage linked service for both the copy source and sink store, named "AzureStorageLinkedService" in the sample*

// 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));

*Add the following code to the****Main****method that creates an****Azure blob dataset****.*

*You define a dataset that represents the data to copy from a source to a sink. This Blob dataset references to the Azure Storage linked service you created in the previous step. The dataset takes a parameter whose value is set in an activity that consumes the dataset. The parameter is used to construct the "folderPath" pointing to where the data resides/stored.*

// 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 = "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"

}

}

}

);

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

## Console.WriteLine(SafeJsonConvert.SerializeObject(blobDataset, client.SerializationSettings));Create a dataset

// 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));

*Add the following code to the****Main****method that creates a****pipeline with a copy activity****.*

*This pipeline contains one activity and takes two parameters - input blob path and output blob path. The values for these parameters are set when the pipeline is triggered/run. The copy activity refers to the same blob dataset created in the previous step as input and output. When the dataset is used as an input dataset, input path is specified. And, when the dataset is used as an output dataset, the output path is specified.*

// 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));

*Add the following code to the****Main****method that****triggers a pipeline run****.*

*This code also sets values of****inputPath****and****outputPath****parameters specified in pipeline with the actual values of source and sink blob paths.*

// 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);

## *Add the following code to the******Main******method to continuously check the status until it finishes copying the data.*

// 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;

## }

## *Add the following code to the******Main******method that retrieves copy activity run details, for example, size of the data read/written.*

// 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();

}

}

## }

## For more information check the below link

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

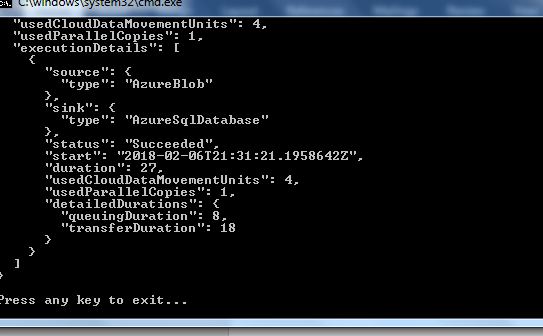
## *Note: Complete code is uploaded on github repository.*

## <https://github.com/dilreddy143/AzureFinalProject1/blob/master/Program.cs>

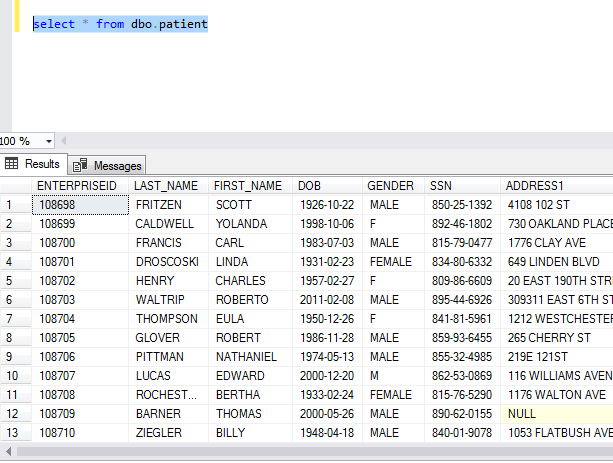
## Run the code

## Build and start the application, then verify the pipeline execution.

Check the console to make sure there are no errors.



Then login to the database and run below queries to and see if the data is loaded in the tables.

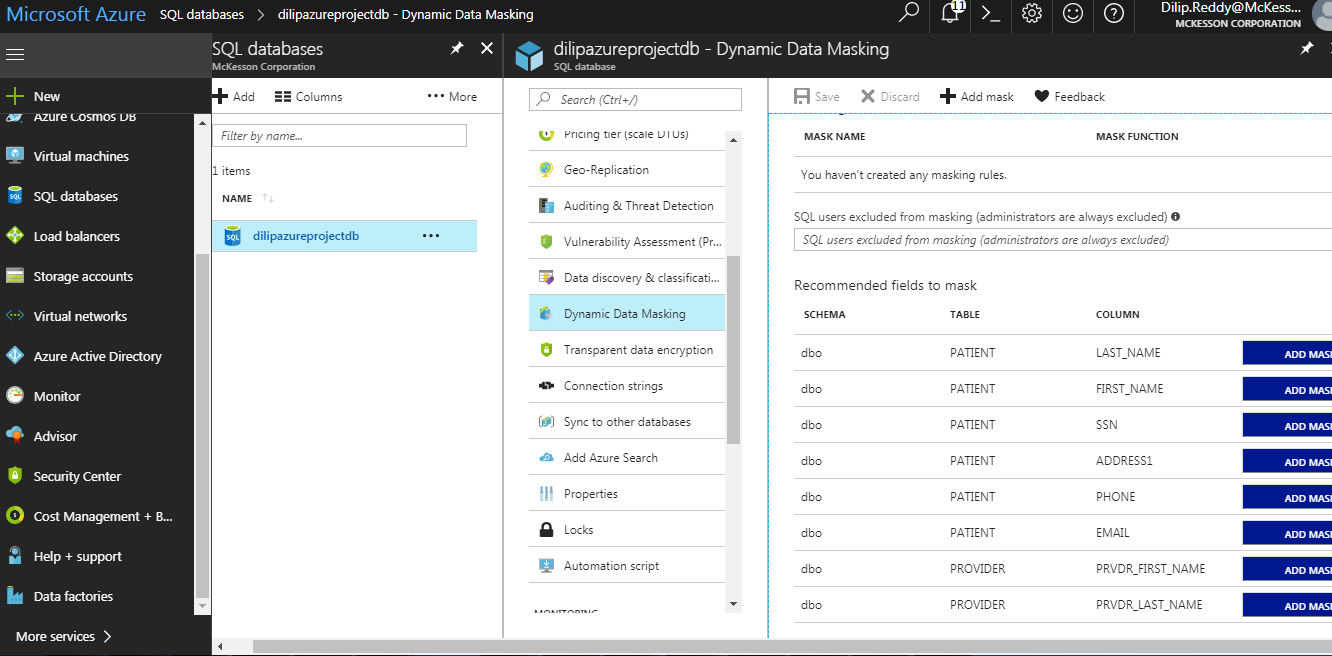


**5.0 Data Masking**

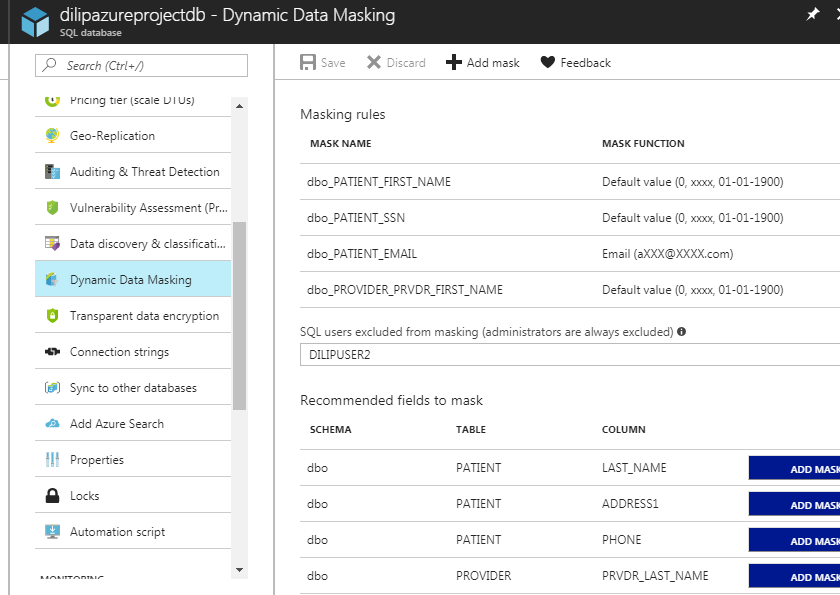
For masking the data

Login to azure portal

Click on SQL databases->Dynamic Data Masking

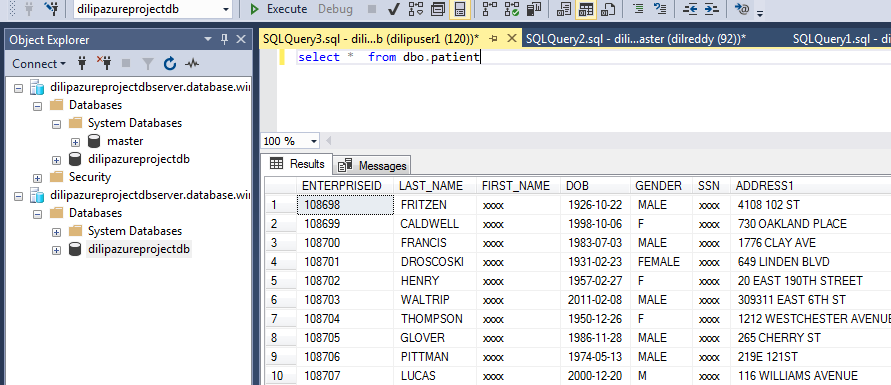


Exclude dilipuser2 from masking and click on ADD MASK for the columns you want to mask and click on save.



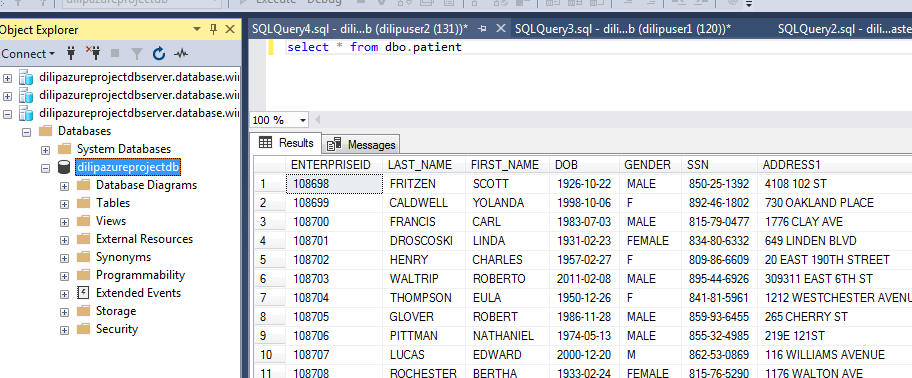
Login to SQL DB as dilipuser1 and check the data.

As shown below, FIRSTNAME, SSN columns are masked.



As we excluded dilipuser2 from masking, the data should not mask for this user.

Login to SQL DB as dilipuser2 and make sure the data is not masked.

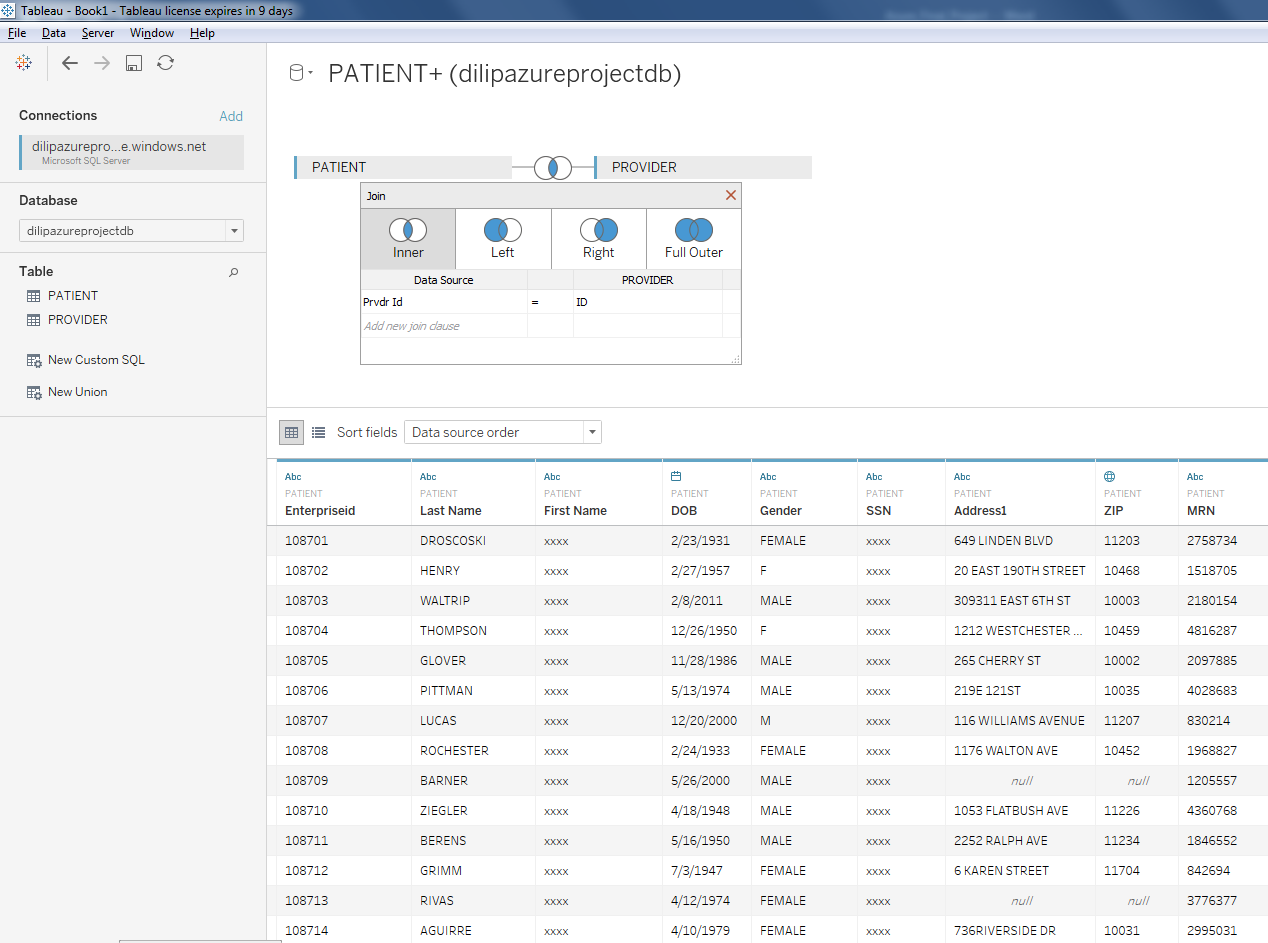


**6.0 Tableau Report**

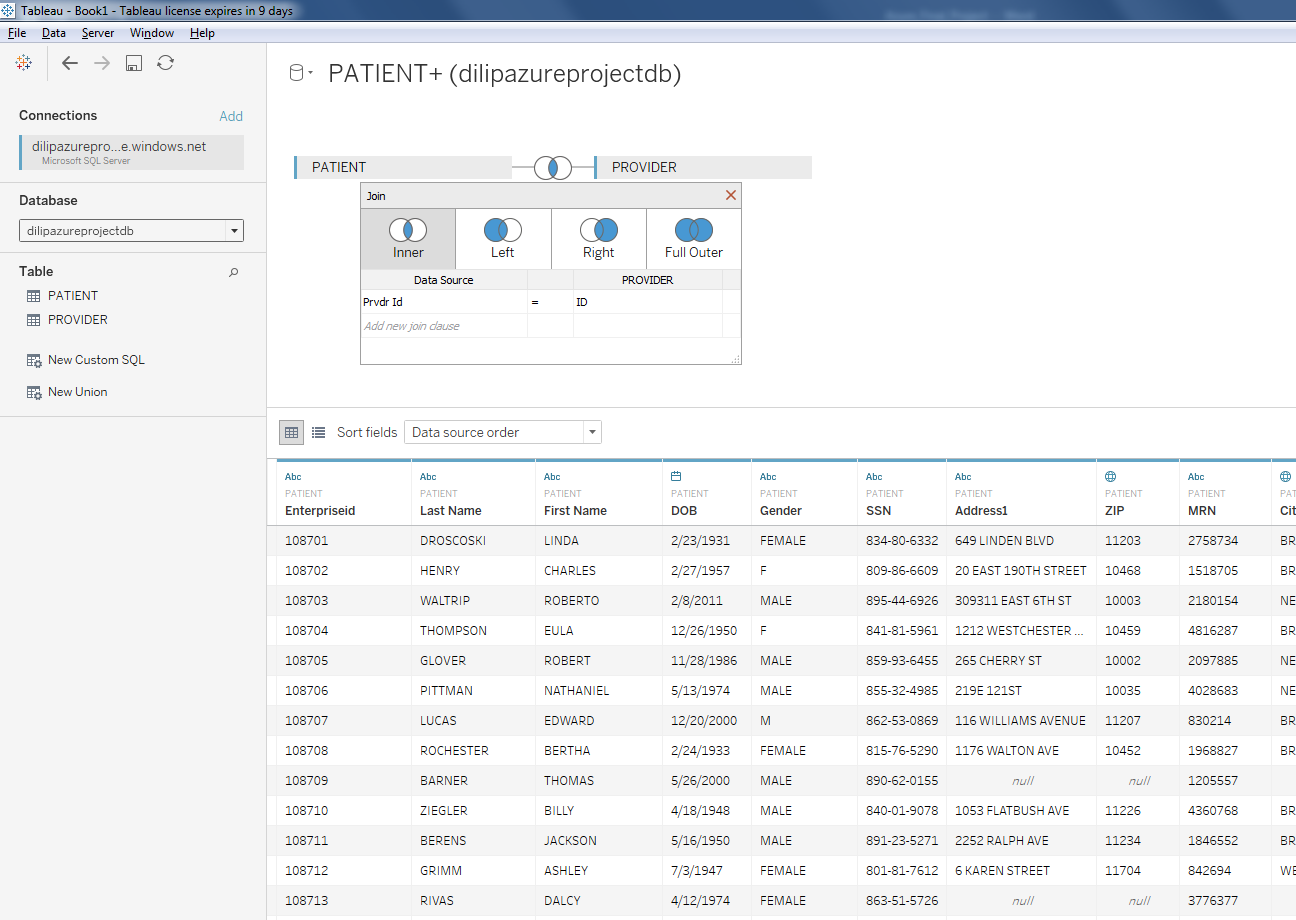
To view the data in report

* 1. Install Tableau trial version
  2. Connect to SQL Server DB with dilipuser1 to see the masked data. If you want to see the unmasked data connect to SQL DB using dilipuser2
  3. Join the provider and patient tables on PRVDR\_ID column
  4. Check the data

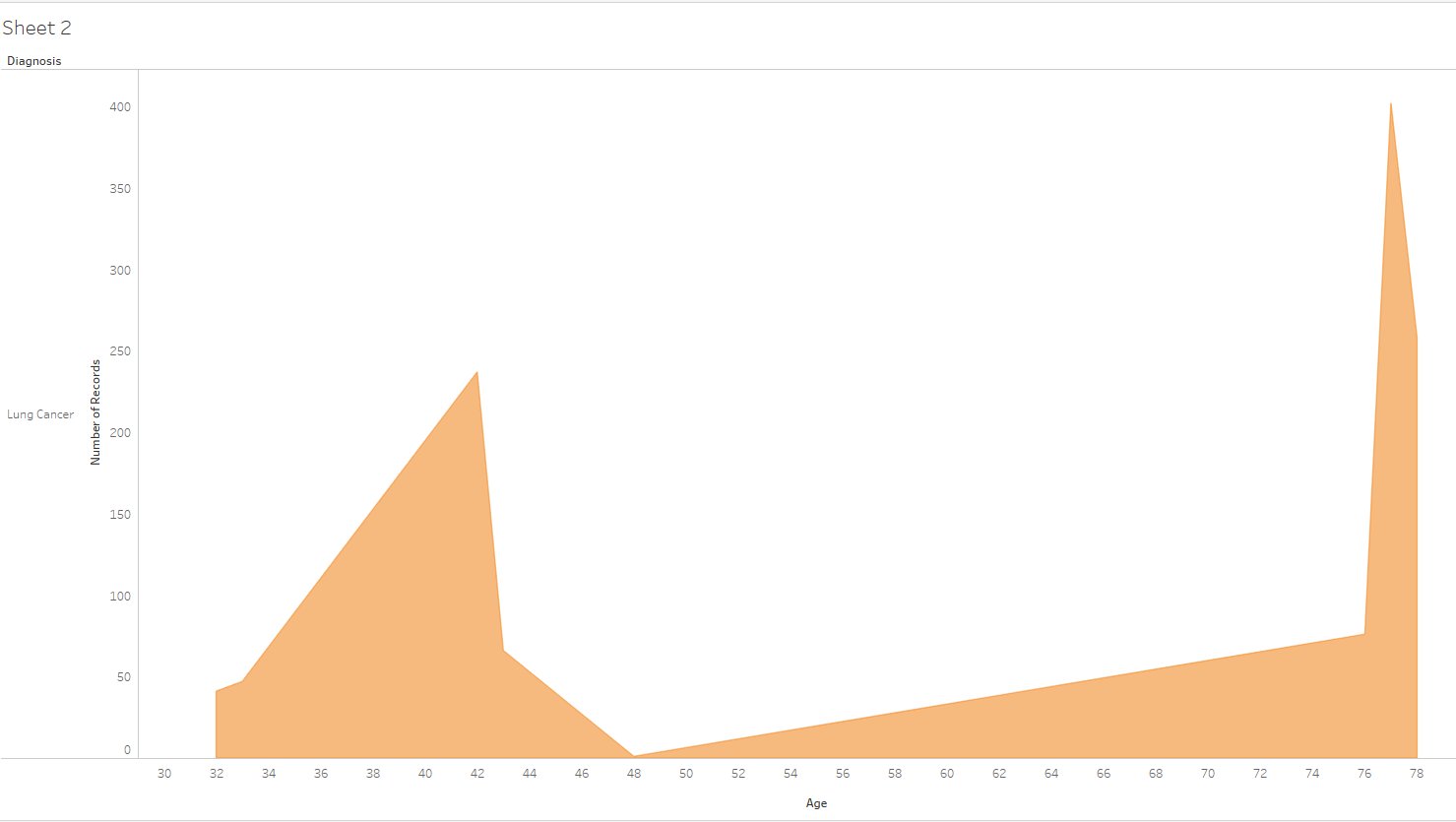
**Masked FIRST NAME and SSN (dilipuser1)**



**Unmasked FIRST NAME and SSN Data (dilipuser2)**



For the same above data set, I created lung cancer graph for each age group in tableau dash board. The below graph shows number of lung cancer patients against a age group.



**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

**Cons**:

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

**What I liked:**

It is easy to mask the data using the dynamic data masking service. The GUI is very user friendly.

**What I did not like:**

* I feel there should be more masking functions.

Example: Mask the zip code and then mask the address based on the zip code, so that city and

zip code will be in sync. Otherwise we can have zip code as 77380 but the city can be Sunnyvale.

* Instead of masking data with xxxx we should mask with some valid names so that users should not know the data is masked.

**Next:**

I would like to run the masking rules on a larger dataset and would like to check the performance.

**Two minute (short):**

<https://youtu.be/6K-Yg_f0LiI>

**15 minutes (long):**

<https://youtu.be/3jxA-Lr5IQU>

**GitHub Repository with all artifacts:**

<https://github.com/dilreddy143/AzureFinalProject>

**Other Links:**

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

<https://azure.microsoft.com/en-us/blog/dynamic-data-masking-generally-available/>