

This PDF contains the contents of folders and files from the directory 'Monitoring.ETL.Domain' and its subdirectories.

App.config (Monitoring.ETL.Domain\App.config):

```
<?xml version="1.0" encoding="utf-8"?>

<configuration>

  <configSections>

    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->

    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />

  </configSections>

  <appSettings>

    <!-- General key for environment. Some ETLS must behave differently because of the environment-->

    <add key="Environment" value="t"/>

    <add key="SprocCommandTimeout" value="300" />

    <add key="smtp1Server" value="dvsmtpl.plex.com" />

    <add key="smtp1Port" value="25" />

    <add key="smtp1EnableSSL" value="false" />

    <!-- override the last load date when first loading Impact analysis-->

    <add key="ImpactAnalysisLastLoadTime" value="2017-12-31 23:59:59"/>

  </appSettings>

  <entityFramework>

    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework" />
```

```

<providers>

  <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices,
EntityFramework.SqlServer" />

</providers>

</entityFramework>

<startup>

  <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />

</startup>

<connectionStrings>

  <!--<add name="Meta_WarehouseEntities"
connectionString="metadata=res://*/Model.Meta_Warehouse.csdl|res://*/Model.Meta_Warehouse.ssdl|res://*/Model.Met
a_Warehouse.msl;provider=System.Data.SqlClient;provider connection string=&quot;data
source=AH_DBPERF_D01\AH_DBPERF_D01;initial catalog=Meta_Warehouse;integrated
security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;;" providerName="System.Data.EntityClient"
/>-->

  <add name="Meta_WarehouseEntities"
connectionString="metadata=res://*/Model.Meta_Warehouse.csdl|res://*/Model.Meta_Warehouse.ssdl|res://*/Model.Met
a_Warehouse.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=localhost,1433;initial
catalog=Meta_Warehouse;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;;"
providerName="System.Data.EntityClient" />

</connectionStrings>

</configuration>

```

ExtractionDelayFactory.cs (Monitoring.ETL.Domain\ApplicationEvent\ExtractionDelayFactory.cs):

```
using System;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.ApplicationEvent
```

```
{
```

```
    public class ExtractionDelayFactory : IExtractionDelayFactory<RabbitMQ.Model.ApplicationEvent>
```

```
    {
```

```
        public async Task Create(ResultSet<RabbitMQ.Model.ApplicationEvent> extracted, CancellationToken
```

```
cancellation_token, IEtlProcessLogger logger)
```

```
        {
```

```
            if (extracted == null || extracted.Results.Count + extracted.Exceptions.Count < 2000)
```

```
            {
```

```
                await Task.Delay(TimeSpan.FromSeconds(5), cancellation_token);
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\ApplicationEvent\RabbitMQExtractor.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.ApplicationEvent
```

```
{
```

```
    public class RabbitMQExtractor : IExtractor<RabbitMQ.Model.ApplicationEvent>
```

```
    {
```

```
        private RabbitMQ.ApplicationEvent.Consumer _eventRepository;
```

```
        public async Task<ResultSet<RabbitMQ.Model.ApplicationEvent>>
```

```
ExtractAllSinceLastExtractAsync(CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
    {
```

```
        _eventRepository = _eventRepository ?? new RabbitMQ.ApplicationEvent.Consumer(logger);
```

```
        var events = await _eventRepository.ExtractAsync();
```

```
        return events;
```

```
    }
```

```
        public Task<IEnumerable<RabbitMQ.Model.ApplicationEvent>> ExtractBetweenAsync(DateTime startDate,
```

```
DateTime endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
    {
```

```
        throw new NotImplementedException();
```

```
    }
```

```
}
```

```
}
```

RabbitMQWarehouseTransformer.cs (Monitoring.ETL.Domain\ApplicationEvent\RabbitMQWarehouseTransformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.ApplicationEvent.Transform
```

```
{
```

```
    public class RabbitMQWarehouseTransformer : ITransformer<RabbitMQ.Model.ApplicationEvent,  
Warehouse.Model.Application_Event_w>
```

```
{
```

```
    public RabbitMQWarehouseTransformer()
```

```
{
```

```
}
```

```
    public async Task<IEnumerable<Warehouse.Model.Application_Event_w>>
```

```
TransformAsync(IEnumerable<RabbitMQ.Model.ApplicationEvent> extracted, CancellationToken cancellationToken,  
IEtlProcessLogger logger)
```

```
{
```

```
await Task.Yield();

// TODO: Move Warehouse_Load_Add here to generate key?

int i = 1;

return extracted.Select(e =>

    new Warehouse.Model.Application_Event_w

    {

        Tags = GetTags(e, i),

        Load_Event_Number = i++,

        Error_Key = e.Error_Key,

        PCN = e.PCN,

        PUN = e.PUN,

        Event_Type = e.Event_Type,

        Message = e.Message,

        Filename = e.Filename,

        Event_Date = e.Timestamp.ToLocalTime(),

        Web_Server = e.Destination_Hostname,

        HTTP_Referer = e.Referer,

        HTTP_User_Agent = e.User_Agent,

        Path_Info = e.Path_Info,

        Request_Method = e.Method,

        //Script_Name = e.Script_Name,

        //Lock_Chain = e.Lock_Chain,

        SQL_Server = e.Sql_Server,

        //Path_Translated = e.Path_Translated,

        //Server_Name = e.Server_Name,

        Element_List = e.Element_List,
```

```
Session_Info = e.Session_Info,

Setting_Group = e.Setting_Group?.Trim(),

Setting_Name = e.Setting_Name?.Trim(),

Session_Id = e.Session_Id?.Trim(),

Exception_Data_Source_Key = e.Exception_Data_Source_Key.TryParseInt(),

Exception_Data_Source_Name = e.Exception_Data_Source_Name?.Trim(),

Exception_Column_Name = e.Exception_Column_Name?.Trim(),

Exception_SQL_Command = e.Exception_SQL_Command,

Exception_Error_Message = e.Exception_Error_Message,

User_Agent_Browser_Name = e.User_Agent_Name?.Trim(),

User_Agent_Browser_Major_Version = e.User_Agent_Major.TryParseInt(),

User_Agent_OS_Name = e.User_Agent_OS?.Trim(),

User_Agent_Device = e.User_Agent_Device?.Trim(),

Missing_Image_Path = e.Missing_Image_Path,

Stack_Trace = e.Stack_Trace,

Exception_Location = e.Exception_Location,

Service_Application_Name = e.ApplicationName,

Referer_Path = e.Referer_Path,

Event_Severity = e.Severity,

Data_Center = e.Data_Center,

Environment = e.Environment,

Node_Id = e.NodeId,

Thread_Id = e.ThreadId,

Previous_Change_Version = e.PreviousChangeVersion

}).ToList();

}
```

```

private IEnumerable<Warehouse.Model.Tag_w> GetTags(RabbitMQ.Model.ApplicationEvent model, int
loadEventNumber)

{

return model.Event_Sub_Type?.Select(

t => new Warehouse.Model.Tag_w

{

Elastic_Id = model.Elastic_Id,

Tag_Name = t,

Load_Event_Number = loadEventNumber

}) ??

new List<Warehouse.Model.Tag_w>();

}

}

```

```

public static class StringExtensions

{

public static int? TryParseInt(this string text)

{

if (int.TryParse(text, out int number))

{

return number;

}

else

{

return null;

}

}

}

```



```

    }

    }

}

```

WarehouseLoader.cs (Monitoring.ETL.Domain\ApplicationEvent\WarehouseLoader.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```
using System.Configuration;
```

```
using System.Data.SqlClient;
```

```
using Monitoring.ETL.Domain.Warehouse.Model;
```

```
namespace Monitoring.ETL.Domain.ApplicationEvent
```

```
{
```

```
    public class WarehouseLoader : Loader<Application_Event_w>
```

```
    {
```

```
        public WarehouseLoader()
```

```
        {
```

```
            //option to load the text data or not.
```

```
            if (!bool.TryParse(ConfigurationManager.AppSettings["Application_Event_LoadAuburnHillsTextData"], out var
loadTextData))
```

```
            {
                loadTextData = true;
```

```
                var textDataParam = new SqlParameter("LoadAuburnHillsTextData", loadTextData);
```

```
                SqlParameter[] ps = { textDataParam };
```

```

        CreateRepository("Fact_Application_Event_Add", ps, true);
    }
}
}

```

AzureQueueCredentialProvider.cs (Monitoring.ETL.Domain\AzureQueue\AzureQueueCredentialProvider.cs):

```

using System;

using Microsoft.Azure.Storage.Auth;

using Plex.Infrastructure.ConfigSystems;

namespace Monitoring.ETL.Domain.AzureQueue
{
    internal sealed class AzureQueueCredentialProvider
    {
        private readonly PlexRegistrySystem _registry;

        public AzureQueueCredentialProvider()
        {
            _registry = new PlexRegistrySystem();
        }

        public StorageCredentials GetCredentials()
        {
            // TODO: This should be driven by consul.

            var accountName = GetRegistryValue("CloudOps", "Monitoring", "AzureQueue", "AccountName");

            var keyValue = GetRegistryValue("CloudOps", "Monitoring", "AzureQueue", "AccessKey");

```

```

        return new StorageCredentials(accountName, keyValue);
    }

    private string GetRegistryValue(string moduleGroup, string module, string keyName, string valueName)
    {
        var value = _registry.GetString(moduleGroup, module, keyName, valueName, null);

        if (string.IsNullOrEmpty(value))
        {
            throw new InvalidOperationException(
                string.Format(
                    "The Warehouse server was not found in the registry. Path: HKLM\\Software\\Plex\\{0}\\{1}\\{2} Value: {3}",
                    moduleGroup,
                    module,
                    keyName,
                    valueName));
        }

        return value;
    }
}

IAzureQueueMessage.cs (Monitoring.ETL.Domain\\AzureQueue\\IAzureQueueMessage.cs):

using ETL.Process;

```

```
namespace Monitoring.ETL.Domain.AzureQueue

{

    public interface IAzureQueueMessage : IExtractModel

    {

        string JsonMessage { get; set; }

    }

}
```

Repository.cs (Monitoring.ETL.Domain\AzureQueue\Repository.cs):

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using Microsoft.Azure.Storage;

using Microsoft.Azure.Storage.Auth;

using Microsoft.Azure.Storage.Queue;
```

```
namespace Monitoring.ETL.Domain.AzureQueue

{

    public class Repository<T>

        where T : IAzureQueueMessage, new()

    {

        private readonly CloudQueue _queue;

        public Repository(string queueName)
```

```

{

    var credentialProvider = new AzureQueueCredentialProvider();

    var storageAccount = new CloudStorageAccount(credentialProvider.GetCredentials(), true);

    var queueClient = storageAccount.CreateCloudQueueClient();

    _queue = queueClient.GetQueueReference(queueName);

}

public async Task<IEnumerable<T>> Get(int count, CancellationToken cancellationToken)

{

    IEnumerable<CloudQueueMessage> messages = new List<CloudQueueMessage>();

    List<T> results = new List<T>();

    messages = await _queue.GetMessagesAsync(count, cancellationToken);

    foreach (var message in messages)

    {

        results.Add(new T { JsonMessage = message.AsString });

        await _queue.DeleteMessageAsync(message, cancellationToken);

    }

    return results;

}

}

```

ClassicDataSource.cs (Monitoring.ETL.Domain\ClassicDataSource\ClassicDataSource.cs):

```

using ETL.Process;

```

```
namespace Monitoring.ETL.Domain.ClassicDataSource
```

```
{  
  
    public class ClassicDataSource : IExtractModel, SqlJson.IJsonExtractModel  
  
    {  
  
        public string JsonMessage { get; set; }  
  
    }  
  
}
```

DelayFactory.cs (Monitoring.ETL.Domain\ClassicDataSource\DelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.ClassicDataSource
```

```
{  
  
    public class DelayFactory : TimeOfDayExtractionDelayFactory<ClassicDataSource>  
  
    {  
  
        protected override TimeSpan TimeOfDay => new TimeSpan(3, 30, 00);  
  
    }  
  
}
```

Extractor.cs (Monitoring.ETL.Domain\ClassicDataSource\Extractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.User;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.ClassicDataSource
```

```
{  
  
    public class Extractor : IExtractor<ClassicDataSource>  
  
    {  
  
        private readonly SqlJson.Repository<ClassicDataSource> _repository;
```

```
        public Extractor()
```

```
        {  
  
            _repository = new SqlJson.Repository<ClassicDataSource>(new List<string> { "vp" },  
  
                "Plexus_Rendering",  
  
                "");  
  
        }
```

```
        public async Task<ResultSet<ClassicDataSource>> ExtractAllSinceLastExtractAsync(CancellationTokencancellationTokentoken, IETLProcessLogger logger)
```

```
        {  
  
            var takeRecords = 5000;  
  
            var currentOffset = 0;  
  
  
            var template = @"
```

USE Plexus_Rendering;

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

SELECT

DS.Datasource_Key,

DS.Datasource_Name,

DST.Datasource_Type,

ISNULL(DSS.Datasource_Status, 'Unknown') AS Data_Source_Status,

DS.Module_Key,

ISNULL(DB.[Database_Name], '') AS Stored_Procedure_Database,

ISNULL(DSP.[Procedure_Name], '') AS Stored_Procedure_Name,

DST.Stored_Procedure_Type,

DST.Class_Type,

ISNULL(DSS.Production, 0) AS Production,

DS.Internal,

DS.Query_Writer_PCN,

DS.PCN

FROM Plexus_Rendering.dbo.Datasource AS DS

LEFT OUTER JOIN Plexus_Rendering.dbo.Datasource_Status AS DSS

ON DSS.Datasource_Status_Key = DS.Datasource_Status_Key

JOIN Plexus_Rendering.dbo.Datasource_Type AS DST

ON DST.Datasource_Type_Key = DS.Datasource_Type_Key

LEFT OUTER JOIN Plexus_Rendering.dbo.Datasource_Sproc AS DSP

ON DSP.Datasource_Key = DS.Datasource_Key

LEFT OUTER JOIN Plexus_System.dbo.Plexus_Database AS DB

ON DB.Plexus_Database_Key = DSP.Database_Key

ORDER BY DS.Datasource_Key

OFFSET {0} ROWS FETCH NEXT {1} ROWS ONLY

FOR JSON PATH;"

```
var resultset = new ResultSet<ClassicDataSource>();
```

```
var loadMore = true;
```

```
while (loadMore)
```

```
{
```

```
    var query = string.Format(template, currentOffset, takeRecords);
```

```
    _repository.SetQueryTemplate(query);
```

```
    var result = await _repository.ExtractAllSinceLastExtractAsync(cancellationToken, logger);
```

```
    if (result.Results.Count == 0)
```

```
    {
```

```
        loadMore = false;
```

```
    }
```

```
    else
```

```
    {
```

```
        currentOffset += takeRecords;
```

```
    }
```

```
    foreach (var r in result.Results)
```

```

    {
        resultset.Results.Add(r);
    }

    foreach (var e in result.Exceptions)
    {
        resultset.Exceptions.Add(e);
    }
}

return resultset;
}

```

```

public Task<IEnumerable<ClassicDataSource>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
Cancellation token cancellationToken, IEtlProcessLogger logger)
{
    throw new NotImplementedException();
}
}
}

```

RabbitMQDelayFactory.cs (Monitoring.ETL.Domain\ClassicDataSource\RabbitMQDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.ClassicDataSource
```

```
{
```

```

public class RabbitMQDelayFactory :
ThresholdExtractionDelayFactory<RabbitMQ.Model.RabbitMQClassicDataSource>

{

    protected override int MaxThresholdForDelay => int.MaxValue;

    protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);

}

}

```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\ClassicDataSource\RabbitMQExtractor.cs):

```

using System;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;


using ETL.Process;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.ClassicDataSource
{

    public sealed class RabbitMQExtractor : IExtractor<RabbitMQ.Model.RabbitMQClassicDataSource>

    {

        private RabbitMQ.ClassicDataSource.Consumer _consumer;


        public async Task<ResultSet<RabbitMQ.Model.RabbitMQClassicDataSource>>

```

ExtractAllSinceLastExtractAsync(CancellationTokentoken cancellationToken, IEtlProcessLogger logger)

```
{  
    _consumer = _consumer ?? new RabbitMQ.ClassicDataSource.Consumer(logger);  
    return await _consumer.ExtractAsync();  
}
```

public Task<IEnumerable<RabbitMQ.Model.RabbitMQClassicDataSource>> ExtractBetweenAsync(DateTime
startDate, DateTime endDate, CancellationTokentoken cancellationToken, IEtlProcessLogger logger)

```
{  
    throw new NotImplementedException();  
}  
}  
}
```

RabbitMQLoader.cs (Monitoring.ETL.Domain\ClassicDataSource\RabbitMQLoader.cs):

namespace Monitoring.ETL.Domain.ClassicDataSource

```
{  
    public class RabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.RabbitMQClassicDataSource>  
    {  
        public RabbitMQLoader() : base(new RabbitMQ.ClassicDataSource.Producer())  
        {  
        }  
    }  
}
```

RabbitMQWarehouseTransformer.cs

(Monitoring.ETL.Domain\ClassicDataSource\RabbitMQWarehouseTransformer.cs):

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.ClassicDataSource

{

public class RabbitMQWarehouseTransformer : ITransformer<RabbitMQ.Model.RabbitMQClassicDataSource,
Warehouse.Model.Classic_Data_Source_w>

{

public async Task<IEnumerable<Warehouse.Model.Classic_Data_Source_w>>

TransformAsync(IEnumerable<RabbitMQ.Model.RabbitMQClassicDataSource> extracted, CancellationToken
cancellationToken, IEtlProcessLogger logger)

{

await Task.Yield();

return extracted.Select(pc =>

new Warehouse.Model.Classic_Data_Source_w

{

JSON_Message = pc.JsonMessage

```

    });

}

}

}

```

Transformer.cs (Monitoring.ETL.Domain\ClassicDataSource\Transformer.cs):

```

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

```

```

using ETL.Process;

```

```

using Monitoring.ETL.Process;

```

```

namespace Monitoring.ETL.Domain.ClassicDataSource

```

```

{

    public class Transformer : ITransformer<ClassicDataSource, RabbitMQ.Model.RabbitMQClassicDataSource>

    {

        public async Task<IEnumerable<RabbitMQ.Model.RabbitMQClassicDataSource>>

```

```

TransformAsync(IEnumerable<ClassicDataSource> extracted, CancellationToken cancellationToken, IEtlProcessLogger
logger)

```

```

    {

        await Task.Yield();

```

```

        return extracted.Select(pc =>

```

```

new RabbitMQ.Model.RabbitMQClassicDataSource

{

    JsonMessage = pc.JsonMessage

});

}

}

}

```

WarehouseLoader.cs (Monitoring.ETL.Domain\ClassicDataSource\WarehouseLoader.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.ClassicDataSource
```

```

{

    public class WarehouseLoader : Loader<Warehouse.Model.Classic_Data_Source_w>

    {

        public WarehouseLoader() : base("Dim_Classic_Data_Source_Add")

        {

        }

    }

}

```

ClassicScreen.cs (Monitoring.ETL.Domain\ClassicScreen\ClassicScreen.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.ClassicScreen
```

```
{  
  
    public class ClassicScreen : IExtractModel, SqlJson.IJsonExtractModel  
  
    {  
  
        public string JsonMessage { get; set; }  
  
    }  
  
}
```

DelayFactory.cs (Monitoring.ETL.Domain\ClassicScreen\DelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.ClassicScreen
```

```
{  
  
    public class DelayFactory : TimeOfDayExtractionDelayFactory<ClassicScreen>  
  
    {  
  
        protected override TimeSpan TimeOfDay => new TimeSpan(3, 30, 00);  
  
    }  
  
}
```

Extractor.cs (Monitoring.ETL.Domain\ClassicScreen\Extractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```



```
namespace Monitoring.ETL.Domain.ClassicScreen

{

    public class PlexusCustomerExtractor : IExtractor<ClassicScreen>

    {

        private readonly SqlJson.Repository<ClassicScreen> _repository;


        public PlexusCustomerExtractor()

        {

            _repository = new SqlJson.Repository<ClassicScreen>(

                new List<string> { "vp" },

                "Plexus_Rendering",

                @"

USE Plexus_Rendering;

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;


SELECT

    S.Screen_Key,

    S.Screen_Name,

    S.Module_Key,

    SS.Screen_Status,

    SS.Production,

    S.Internal,

    S.Add_Date

FROM Plexus_Rendering.dbo.Screen AS S

JOIN Plexus_Rendering.dbo.Screen_Status AS SS
```

```
ON SS.Screen_Status_Key = S.Screen_Status_Key
```

```
FOR JSON PATH;");
```

```
}
```

```
public Task<ResultSet<ClassicScreen>> ExtractAllSinceLastExtractAsync(CancellationTokentoken,
IETLProcessLogger logger)
```

```
{
```

```
    return _repository.ExtractAllSinceLastExtractAsync(cancellationTokentoken, logger);
```

```
}
```

```
public Task<IEnumerable<ClassicScreen>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
CancellationTokentoken, IETLProcessLogger logger)
```

```
{
```

```
    throw new NotImplementedException();
```

```
}
```

```
}
```

```
}
```

RabbitMQDelayFactory.cs (Monitoring.ETL.Domain\ClassicScreen\RabbitMQDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.ClassicScreen
```

```
{
```

```
    public class RabbitMQDelayFactory : ThresholdExtractionDelayFactory<RabbitMQ.Model.RabbitMQClassicScreen>
```

```
    {
```

```
        protected override int MaxThresholdForDelay => int.MaxValue;
```

```

        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);
    }
}

```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\ClassicScreen\RabbitMQExtractor.cs):

```

using System;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.ClassicScreen
{
    public sealed class RabbitMQExtractor : IExtractor<RabbitMQ.Model.RabbitMQClassicScreen>
    {
        private RabbitMQ.ClassicScreen.Consumer _consumer;

        public async Task<ResultSet<RabbitMQ.Model.RabbitMQClassicScreen>>
ExtractAllSinceLastExtractAsync(CancellationToken cancellationToken, IEtlProcessLogger logger)
        {
            _consumer = _consumer ?? new RabbitMQ.ClassicScreen.Consumer(logger);

            return await _consumer.ExtractAsync();
        }
    }
}

```

```
public Task<IEnumerable<RabbitMQ.Model.RabbitMQClassicScreen>> ExtractBetweenAsync(DateTime startDate,
DateTime endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)

{

    throw new NotImplementedException();

}

}

}
```

RabbitMQLoader.cs (Monitoring.ETL.Domain\ClassicScreen\RabbitMQLoader.cs):

```
namespace Monitoring.ETL.Domain.ClassicScreen

{

    public class RabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.RabbitMQClassicScreen>

    {

        public RabbitMQLoader()

            : base(new RabbitMQ.ClassicScreen.Producer())

        {

        }

    }

}
```

RabbitMQWarehouseTransformer.cs (Monitoring.ETL.Domain\ClassicScreen\RabbitMQWarehouseTransformer.cs):

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;
```

```

using ETL.Process;

using Monitoring.ETL.Process;


namespace Monitoring.ETL.Domain.ClassicScreen
{

    public class RabbitMQWarehouseTransformer : ITransformer<RabbitMQ.Model.RabbitMQClassicScreen,
Warehouse.Model.Classic_Screen_w>
    {

        public async Task<IEnumerable<Warehouse.Model.Classic_Screen_w>>
TransformAsync(IEnumerable<RabbitMQ.Model.RabbitMQClassicScreen> extracted, CancellationToken
cancellationToken, IEtlProcessLogger logger)
        {

            await Task.Yield();


            return extracted.Select(pc =>
                new Warehouse.Model.Classic_Screen_w
                {

                    JSON_Message = pc.JsonMessage

                });
        }
    }
}

```

Transformer.cs (Monitoring.ETL.Domain\ClassicScreen\Transformer.cs):

```

using System.Collections.Generic;

using System.Linq;

```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.ClassicScreen
```

```
{
```

```
    public class Transformer : ITransformer<ClassicScreen, RabbitMQ.Model.RabbitMQClassicScreen>
```

```
    {
```

```
        public async Task<IEnumerable<RabbitMQ.Model.RabbitMQClassicScreen>>
```

```
TransformAsync(IEnumerable<ClassicScreen> extracted, CancellationToken cancellationToken, IEtlProcessLogger  
logger)
```

```
    {
```

```
        await Task.Yield();
```

```
        return extracted.Select(pc =>
```

```
            new RabbitMQ.Model.RabbitMQClassicScreen
```

```
            {
```

```
                JsonMessage = pc.JsonMessage
```

```
            });
```

```
        }
```

```
    }
```

```
}
```

WarehouseLoader.cs (Monitoring.ETL.Domain\ClassicScreen\WarehouseLoader.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.ClassicScreen
```

```
{  
  
    public class WarehouseLoader : Loader<Warehouse.Model.Classic_Screen_w>  
  
    {  
  
        public WarehouseLoader()  
  
            : base("Dim_Classic_Screen_Add")  
  
        {  
  
        }  
  
    }  
  
}
```

CloudDataSource.cs (Monitoring.ETL.Domain\CloudDataSource\CloudDataSource.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.CloudDataSource
```

```
{  
  
    public class CloudDataSource : IExtractModel, SqlJson.IJsonExtractModel  
  
    {  
  
        public string JsonMessage { get; set; }  
  
    }  
  
}
```

DelayFactory.cs (Monitoring.ETL.Domain\CloudDataSource\DelayFactory.cs):

```
using System;
```

```

namespace Monitoring.ETL.Domain.CloudDataSource

{

    public class DelayFactory : TimeOfDayExtractionDelayFactory<CloudDataSource>

    {

        protected override TimeSpan TimeOfDay => new TimeSpan(3, 30, 00);

    }

}

```

Extractor.cs (Monitoring.ETL.Domain\CloudDataSource\Extractor.cs):

```

using System;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

```

```

namespace Monitoring.ETL.Domain.CloudDataSource

{

    public class Extractor : IExtractor<CloudDataSource>

    {

        private readonly SqlJson.Repository<CloudDataSource> _repository;

        public Extractor()

        {

            _repository = new SqlJson.Repository<CloudDataSource>(

```



```

        new List<string> { "n1" },

        "Cloud",

        @"

USE Cloud;

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

SELECT

    DS.Data_Source_Key,

    DS.Data_Source_Name,

    DST.Data_Source_Type,

    DS.Module_Key,

    DS.[Database_Name],

    DS.Stored_Procedure_Name,

    DS.Internal,

    DS.Owner_PCN

FROM dbo.Data_Source AS DS

JOIN dbo.Data_Source_Type AS DST

    ON DST.Data_Source_Type_Key = DS.Data_Source_Type_Key

FOR JSON PATH;");

    }

```

```

        public Task<ResultSet<CloudDataSource>> ExtractAllSinceLastExtractAsync(Cancellation_token cancellationToken,

IEtlProcessLogger logger)

        {

            return _repository.ExtractAllSinceLastExtractAsync(cancellationToken, logger);

        }

```

```

    public Task<IEnumerable<CloudDataSource>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
CancellationTokentoken, IETLProcessLogger logger)

    {

        throw new NotImplementedException();

    }

}

}

```

RabbitMQDelayFactory.cs (Monitoring.ETL.Domain\CloudDataSource\RabbitMQDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.CloudDataSource
```

```

{

    public class RabbitMQDelayFactory :

ThresholdExtractionDelayFactory<RabbitMQ.Model.RabbitMQCloudDataSource>

    {

        protected override int MaxThresholdForDelay => int.MaxValue;

        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);

    }

}

```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\CloudDataSource\RabbitMQExtractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.CloudDataSource
```

```
{
```

```
    public sealed class RabbitMQExtractor : IExtractor<RabbitMQ.Model.RabbitMQCloudDataSource>
```

```
    {
```

```
        private RabbitMQ.CloudDataSource.Consumer _consumer;
```

```
        public async Task<ResultSet<RabbitMQ.Model.RabbitMQCloudDataSource>>
```

```
ExtractAllSinceLastExtractAsync(CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
    {
```

```
        _consumer = _consumer ?? new RabbitMQ.CloudDataSource.Consumer(logger);
```

```
        return await _consumer.ExtractAsync();
```

```
    }
```

```
        public Task<IEnumerable<RabbitMQ.Model.RabbitMQCloudDataSource>> ExtractBetweenAsync(DateTime
```

```
startDate, DateTime endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
    {
```

```
        throw new NotImplementedException();
```

```
    }
```

```
}
```

```
}
```

RabbitMQLoader.cs (Monitoring.ETL.Domain\CloudDataSource\RabbitMQLoader.cs):

```
namespace Monitoring.ETL.Domain.CloudDataSource
```

```
{  
  
    public class RabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.RabbitMQCloudDataSource>  
  
    {  
  
        public RabbitMQLoader()  
  
            : base(new RabbitMQ.CloudDataSource.Producer())  
  
        {  
  
        }  
  
    }  
  
}
```

RabbitMQWarehouseTransformer.cs (Monitoring.ETL.Domain\CloudDataSource\RabbitMQWarehouseTransformer.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.CloudDataSource
```

```
{  
  
    public class RabbitMQWarehouseTransformer : ITransformer<RabbitMQ.Model.RabbitMQCloudDataSource,  
Warehouse.Model.Cloud_Data_Source_w>  
  
    {
```

```

        public async Task<IEnumerable<Warehouse.Model.Cloud_Data_Source_w>>
TransformAsync(IEnumerable<RabbitMQ.Model.RabbitMQCloudDataSource> extracted, CancellationToken
cancellationToken, IEtlProcessLogger logger)
    {
        await Task.Yield();

        return extracted.Select(pc =>
            new Warehouse.Model.Cloud_Data_Source_w
            {
                JSON_Message = pc.JsonMessage
            });
    }
}

```

Transformer.cs (Monitoring.ETL.Domain\CloudDataSource\Transformer.cs):

```

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.CloudDataSource
{
    public class Transformer : ITransformer<CloudDataSource, RabbitMQ.Model.RabbitMQCloudDataSource>

```

```

{

    public async Task<IEnumerable<RabbitMQ.Model.RabbitMQCloudDataSource>>

TransformAsync(IEnumerable<CloudDataSource> extracted, CancellationToken cancellationToken, IEtlProcessLogger

logger)

    {

        await Task.Yield();

        return extracted.Select(pc =>

            new RabbitMQ.Model.RabbitMQCloudDataSource

            {

                JsonMessage = pc.JsonMessage

            });

        }

    }

}

```

WarehouseLoader.cs (Monitoring.ETL.Domain\CloudDataSource\WarehouseLoader.cs):

```

using ETL.Process;

using Monitoring.ETL.Domain.Warehouse;

namespace Monitoring.ETL.Domain.CloudDataSource

{

    public class WarehouseLoader : Loader<Warehouse.Model.Cloud_Data_Source_w>

    {

        public WarehouseLoader()

            : base("Dim_Cloud_Data_Source_Add")

    }

}

```

```
{  
  
}  
  
}  
  
}
```

ConsulExtractor.cs (Monitoring.ETL.Domain\Consul\ConsulExtractor.cs):

```
using System;  
  
using System.Collections.Generic;  
  
using System.Threading;  
  
using System.Threading.Tasks;  
  
using ETL.Process;  
  
using Monitoring.ETL.Process;  
  
  
namespace Monitoring.ETL.Domain.Consul  
{  
  
    public abstract class ConsulExtractor<T> where T : IExtractModel  
  
    {  
  
        public virtual async Task<ResultSet<T>> ExtractAllSinceLastExtractAsync(Cancellation token cancellationToken,  
IEtlProcessLogger logger)  
  
        {  
  
            var resultSet = new ResultSet<T>();  
  
  
  
            try  
  
            {  
  
                var consulHttpClient = new ConsulHttpClient(logger);  
  
                var models = GetModels(consulHttpClient, out var exceptions);
```

```
foreach (var model in models)

{

    resultSet.Results.Add(model);

}


if (exceptions != null)

{

    foreach (var exception in exceptions)

    {

        resultSet.Exceptions.Add(exception);

    }

}


logger.Information($"Total time spent on web requests: {consulHttpClient.TotalRequestDurationMs}");

logger.Information($"Web Request Count: {consulHttpClient.RequestCount}");


logger.Debug(consulHttpClient.GetRequestStatsByBaseUrl());

}

catch (Exception e)

{

    resultSet.Exceptions.Add(e);

}


return resultSet;

}
```



```
public Task<IEnumerable<T>> ExtractBetweenAsync(DateTime startDate, DateTime endDate, CancellationToken  
cancellationToken, IEtlProcessLogger logger)
```

```
{  
    throw new NotImplementedException();  
}
```

```
protected abstract IEnumerable<T> GetModels(ConsulHttpClient consulHttpClient, out IList<Exception>  
exceptions);  
  
}
```

ConsulHttpClient.cs (Monitoring.ETL.Domain\Consul\ConsulHttpClient.cs):

```
using System;  
  
using System.Collections.Generic;  
  
using System.IO;  
  
using System.Linq;  
  
using System.Net;  
  
using System.Security.Cryptography.X509Certificates;  
  
  
using Libraries.Common.Exceptions;  
  
  
using Monitoring.ETL.Domain.Consul.Model;  
  
using Monitoring.ETL.Process;  
  
  
using Newtonsoft.Json;
```

```
using Newtonsoft.Json.Linq;
```

```
using Plex.Infrastructure.ConfigSystems;
```

```
namespace Monitoring.ETL.Domain.Consul
```

```
{
```

```
    public sealed class ConsulHttpClient
```

```
    {
```

```
        private readonly IEtlProcessLogger _logger;
```

```
        private readonly Dictionary<string, string> _consulEnvironmentApiUrls;
```

```
        private readonly string _consulCertPath;
```

```
        private readonly string _consulCertPassword;
```

```
        private readonly List<ApiRequestLog> _apiRequests = new List<ApiRequestLog>();
```

```
        private X509Certificate2 _certificate;
```

```
        private List<DataCenterModel> _dataCenters;
```

```
        public ConsulHttpClient(IEtlProcessLogger logger)
```

```
        {
```

```
            _logger = logger;
```

```
            var registry = new PlexRegistrySystem();
```

```
            _consulCertPath = registry.GetString("CloudOps", "Monitoring", "Consul", "ApiCertPath", string.Empty);
```

```
            _consulCertPassword = registry.GetString("CloudOps", "Monitoring", "Consul", "ApiCertPassword",
```

```
            string.Empty);
```

```

_consulEnvironmentApiUrls = new Dictionary<string, string>(StringComparer.OrdinalIgnoreCase);

var environments = registry.GetStrings("CloudOps", "Monitoring", "Consul", "Environments", null);

if (environments == null || environments.Length == 0)
{
    throw new NullReferenceException("The Environments value cannot be null or empty. The value is pulled
from the registry at HKLM\\Software\\Plex\\CloudOps\\Monitoring\\Consul\\Environments");
}

foreach (var environment in environments)
{
    var registryKey = "ApiUrl-" + environment;

    var environmentApiUrl = registry.GetString("CloudOps", "Monitoring", "Consul", registryKey, string.Empty);

    if (string.IsNullOrEmpty(environmentApiUrl))
    {
        throw new StringNullOrEmptyException("The environment's ApiUrl value cannot be null or empty. The
value is pulled from the registry at HKLM\\Software\\Plex\\CloudOps\\Monitoring\\Consul\\" + registryKey);
    }

    _consulEnvironmentApiUrls.Add(environment, environmentApiUrl);
}

if (string.IsNullOrEmpty(_consulCertPath))

```

```

{
    throw new StringNullOrEmptyException("The ApiCertPath value cannot be null or empty. The value is pulled
from the registry at HKLM\\Software\\Plex\\CloudOps\\Monitoring\\Consul\\ApiCertPath");
}

if (File.Exists(_consulCertPath) == false)
{
    throw new FileNotFoundException($"The consul cert path specified by the registry could not be found or is not
accessible. ({_consulCertPath}) The value is pulled from the registry at
HKLM\\Software\\Plex\\CloudOps\\Monitoring\\Consul\\ApiCertPath");
}
}

internal int RequestCount => _apiRequests.Count;

internal double TotalRequestDurationMs => _apiRequests.Sum(l => l.RequestDuration.TotalMilliseconds);

public IEnumerable<ServiceNodeMappingModel> GetServiceNodeMappings(out IList<Exception> exceptions)
{
    var exceptionsInternal = new List<Exception>();

    var healthChecks = GetPassingServiceHealthChecks(out var getPassingServiceHealthChecksExceptions);
    exceptionsInternal.AddRange(getPassingServiceHealthChecksExceptions);

    var nodes = GetNodes(out var getNodesExceptions).ToList();
    exceptionsInternal.AddRange(getNodesExceptions);
}

```

```
var serviceNodeMappings = new List<ServiceNodeMappingModel>();

foreach (var healthCheck in healthChecks)
{
    var matchingNodes = nodes.Where(n => n.Environment == healthCheck.Environment && n.DataCenter ==
healthCheck.DataCenter && n.Name == healthCheck.Node);

    foreach (var node in matchingNodes)
    {
        try
        {
            // This is being lazy loaded here to reduce API calls.
            if (node.ServiceAddressHostNames == null)
            {
                GetNodeServices(node);
            }

            var serviceNodeMapping = new ServiceNodeMappingModel
            {
                DataCenter = healthCheck.DataCenter,
                Environment = healthCheck.Environment,
                ServiceName = healthCheck.ServiceName,

                // Ordering here just to make sure we get a consistent value in the final JSON .
                ServiceTags = healthCheck.ServiceTags.OrderBy(s => s).ToList()
            };
        }
    }
}
```

```
// The service address/host name can be different than the node's address.
```

```
// This is the case for windows failover clusters. The service address is actually the failover cluster application.
```

```
if (node.ServiceAddressHostNames != null &&
```

```
node.ServiceAddressHostNames.TryGetValue(healthCheck.ServiceName, out var serviceHostNamePort))
```

```
{
```

```
    serviceNodeMapping.NodeName = serviceHostNamePort.HostName ?? node.Name;
```

```
    serviceNodeMapping.Port = serviceHostNamePort.Port;
```

```
}
```

```
else
```

```
{
```

```
    serviceNodeMapping.NodeName = node.Name;
```

```
}
```

```
serviceNodeMapping.NodeName = SanitizeHostName(serviceNodeMapping.NodeName);
```

```
serviceNodeMappings.Add(serviceNodeMapping);
```

```
}
```

```
catch (Exception e)
```

```
{
```

```
    _logger.Information($"Exception while gathering service-node mappings for the node {node.Name} in the {node.DataCenter} data center for the {healthCheck.Environment} environment.");
```

```
    e.Data["Node-Name"] = node.Name;
```

```
    e.Data["Node-Data-Center"] = node.DataCenter;
```

```
    e.Data["Node-Id"] = node.Id;
```

```

        e.Data["Environment"] = healthCheck.Environment;

        e.Data["Service-Name"] = healthCheck.ServiceName;

        exceptionsInternal.Add(e);
    }
}
}

```

```

exceptions = exceptionsInternal;

return serviceNodeMappings;
}

```

```

private IList<DataCenterModel> GetDataCenters(out IList<Exception> exceptions)

```

```

{
    exceptions = new List<Exception>();

```

```

    if (_dataCenters == null)

```

```

    {
        var dataCenters = new List<DataCenterModel>();

```

```

        foreach (var consulEnvironmentApiUrl in _consulEnvironmentApiUrls)

```

```

        {
            try

```

```

            {

```

```

                var environmentDataCenterJson = ExecuteWebRequest(consulEnvironmentApiUrl.Value +
"/v1/catalog/datacenters");

```

```

                var environmentDataCenter =

```

```
JsonConvert.DeserializeObject<List<string>>(environmentDataCenterJson);
```

```
    foreach (var dataCenterName in environmentDataCenter)
```

```
    {
```

```
        dataCenters.Add(
```

```
            new DataCenterModel
```

```
            {
```

```
                Environment = consulEnvironmentApiUrl.Key,
```

```
                Name = dataCenterName
```

```
            });
```

```
    }
```

```
}
```

```
catch (Exception ex)
```

```
{
```

```
    _logger.Information($"Exception while gathering data centers for the {consulEnvironmentApiUrl.Key}
```

```
environment.");
```

```
    ex.Data["Target-Environment"] = consulEnvironmentApiUrl.Key;
```

```
    exceptions.Add(ex);
```

```
}
```

```
}
```

```
    _dataCenters = dataCenters;
```

```
}
```

```
return _dataCenters;
```

```
}
```



```

public IEnumerable<KeyValueModel> GetKeyValuePairs(out IList<Exception> exceptions)
{
    var keyValuePairs = new List<KeyValueModel>();

    var exceptionsInternal = new List<Exception>();

    var dataCenters = GetDataCenters(out var getDataCentersExceptions);

    exceptionsInternal.AddRange(getDataCentersExceptions);

    foreach (var dataCenterModel in dataCenters)
    {
        var baseUrl = _consulEnvironmentApiUrls[dataCenterModel.Environment];

        try
        {
            var keyValueJson = ExecuteWebRequest($"{baseUrl}/v1/kv/?dc={dataCenterModel.Name}&recurse=true");

            var dataCenterKeyValuePairs = JsonConvert.DeserializeObject<List<KeyValueModel>>(keyValueJson);

            dataCenterKeyValuePairs.ForEach(p =>
            {
                if (p.Value != null)
                {
                    p.Value = System.Text.Encoding.UTF8.GetString(Convert.FromBase64String(p.Value));
                }

                p.Environment = dataCenterModel.Environment;
            });
        }
    }
}

```

```

        p.DataCenter = dataCenterModel.Name;

    });

    keyValuePairs.AddRange(dataCenterKeyValuePairs);
}

catch (Exception e)
{
    _logger.Information($"Exception while gathering key value pairs in the {dataCenterModel.Name} data
center for the {dataCenterModel.Environment} environment.");

    e.Data["Target-Environment"] = dataCenterModel.Environment;

    e.Data["Target-Data-Center"] = dataCenterModel.Name;

    exceptionsInternal.Add(e);
}
}

exceptions = exceptionsInternal;

return keyValuePairs;
}

private IEnumerable<HealthCheckModel> GetPassingServiceHealthChecks(out List<Exception> exceptions)
{
    var serviceHealthChecks = new List<HealthCheckModel>();

    exceptions = new List<Exception>();

    var dataCenters = GetDataCenters(out var getDataCentersExceptions);

    exceptions.AddRange(getDataCentersExceptions);

```

```
foreach (var dataCenterModel in dataCenters)

{

    try

    {

        var baseUrl = _consulEnvironmentApiUrls[dataCenterModel.Environment];


        var serviceHealthCheckJson =

ExecuteWebRequest($"{baseUrl}/v1/health/state/passing?dc={dataCenterModel.Name}");


        var dataCenterServiceHealthChecks =

JsonConvert.DeserializeObject<IEnumerable<HealthCheckModel>>(serviceHealthCheckJson);


        dataCenterServiceHealthChecks = dataCenterServiceHealthChecks.Where(h =>

h.CheckId.Equals("serfHealth", StringComparison.OrdinalIgnoreCase) == false);


        dataCenterServiceHealthChecks = dataCenterServiceHealthChecks.Where(h =>

string.IsNullOrEmpty(h.ServiceName) == false);


        foreach (var dataCenterServiceHealthCheck in dataCenterServiceHealthChecks)

        {

            dataCenterServiceHealthCheck.Environment = dataCenterModel.Environment;

            dataCenterServiceHealthCheck.DataCenter = dataCenterModel.Name;


            serviceHealthChecks.Add(dataCenterServiceHealthCheck);

        }

    }

}
```

```

        catch (Exception e)

        {

            _logger.Information($"Exception while gathering passing service checks in the {dataCenterModel.Name}
data center for the {dataCenterModel.Environment} environment.");

            e.Data["Environment"] = dataCenterModel.Environment;

            e.Data["Data-Center"] = dataCenterModel.Name;

            exceptions.Add(e);

        }

    }

```

```

        return serviceHealthChecks;
    }

```

```

private List<NodeModel> GetNodes(out List<Exception> exceptions)
{
    var nodes = new List<NodeModel>();

    exceptions = new List<Exception>();

    var dataCenters = GetDataCenters(out var getDataCentersExceptions);

    exceptions.AddRange(getDataCentersExceptions);

    foreach (var dataCenterModel in dataCenters)
    {

        var baseUrl = _consulEnvironmentApiUrls[dataCenterModel.Environment];

        try

```

```

{

    var nodeJson = ExecuteWebRequest($"{baseUrl}/v1/catalog/nodes?dc={dataCenterModel.Name}");

    var dataCenterNodes = JsonConvert.DeserializeObject<List<NodeModel>>(nodeJson);

    dataCenterNodes.ForEach(n =>

    {

        n.Environment = dataCenterModel.Environment;

        n.DataCenter = dataCenterModel.Name;

    });

    nodes.AddRange(dataCenterNodes);

}

catch (Exception e)

{

    _logger.Information($"Exception while gathering nodes in the {dataCenterModel.Name} data center for the

{dataCenterModel.Environment} environment.");

    e.Data["Environment"] = dataCenterModel.Environment;

    e.Data["Data-Center"] = dataCenterModel.Name;

    exceptions.Add(e);

}

}

return nodes;

}

```

```
private void GetNodeServices(NodeModel nodeModel)

{

    var baseUrl = _consulEnvironmentApiUrls[nodeModel.Environment];


    var singleNodeJson =

ExecuteWebRequest($"{baseUrl}/v1/catalog/node/{nodeModel.Name}?dc={nodeModel.DataCenter}");


    var servicesJToken = JObject.Parse(singleNodeJson).SelectToken("$.Services");


    if (servicesJToken != null)

    {

        var services = servicesJToken.ToObject<Dictionary<string, ServiceModel>>();


        var serviceAddressHostNames = new Dictionary<string, HostNamePortModel>();


        foreach (var serviceKvp in services)

        {

            var serviceModel = serviceKvp.Value;


            var hostNamePortModel = new HostNamePortModel();


            if (string.IsNullOrEmpty(serviceModel.Address) == false)

            {

                var hostName = ReverseDnsLookup(serviceModel.Address);


                if (string.IsNullOrEmpty(hostName) == false)
```

```
{  
  
    hostNamePortModel.HostName = SanitizeHostName(hostName);  
  
}  
  
}
```

```
hostNamePortModel.Port = serviceModel.Port;
```

```
// As long as we got either a port number or a host name, add it to the collection.
```

```
// Redis services are examples of entries that end up getting a port but no host.
```

```
if (hostNamePortModel.Port.HasValue || string.IsNullOrEmpty(hostNamePortModel.HostName) == false)
```

```
{  
  
    serviceAddressHostNames.Add(serviceKvp.Key, hostNamePortModel);  
  
}  
  
}
```

```
nodeModel.ServiceAddressHostNames = serviceAddressHostNames;
```

```
}
```

```
}
```

```
private string ExecuteWebRequest(string url)
```

```
{  
  
    if (_certificate == null)  
  
    {  
  
        _certificate = new X509Certificate2(_consulCertPath, _consulCertPassword);  
  
        ServicePointManager.CheckCertificateRevocationList = false;  
  
        ServicePointManager.ServerCertificateValidationCallback = (a, b, c, d) => true;
```

```
ServicePointManager.Expect100Continue = true;

}

try

{

    var request = (HttpWebRequest)WebRequest.Create(url);

    request.PreAuthenticate = true;

    request.AllowAutoRedirect = true;

    request.ClientCertificates.Add(_certificate);

    request.Credentials = CredentialCache.DefaultCredentials;

    _logger.Debug($"Web Request: {url}");

    var apiRequestLog = new ApiRequestLog()

    {

        Url = request.RequestUri

    };

    _apiRequests.Add(apiRequestLog);

    var startTime = DateTime.Now;

    var response = request.GetResponse();

    var responseStream = response.GetResponseStream();
```



```
        if (responseStream == null)

        {

            throw new NullReferenceException("The response stream was null. Unable to complete request. Request

Url: " + request.RequestUri.ToString());

        }

        string responseString;

        using (responseStream)

        {

            using (var reader = new StreamReader(responseStream))

            {

                responseString = reader.ReadToEnd();

            }

        }

        apiRequestLog.RequestDuration = DateTime.Now.Subtract(startTime);

        return responseString;

    }

    catch (Exception e)

    {

        e.Data["Request-Url"] = url;

        throw;

    }

}
```

```

    }

}

public string GetRequestStatsByBaseUrl()

{
    var stats = _apiRequests.GroupBy(l => l.Url.Host, g => g, (k, e) => new { Host = k, Count = e.Count(),
TotalRequestDurationMs = e.Sum(r => r.RequestDuration.TotalMilliseconds) });

    return string.Join(Environment.NewLine, stats.Select(s => $"{s.Host} - Count: {s.Count} Duration Ms:
{s.TotalRequestDurationMs}"));
}

/// <summary>

/// remove domain qualifications if they exist. We just want host name.

/// </summary>

/// <param name="hostName"></param>

/// <returns></returns>

private static string SanitizeHostName(string hostName)

{
    if (hostName != null && hostName.IndexOf('.') >= 0)

    {
        hostName = hostName.Substring(0, hostName.IndexOf('.'));
    }

    return hostName;
}

```

```
private static string ReverseDnsLookup(string ipAddress)

{

    try

    {

        var hostEntry = Dns.GetHostEntry(ipAddress);

        return hostEntry.HostName;

    }

    catch (Exception)

    {

        return null;

    }

}
```

```
private sealed class ApiRequestLog

{

    public Uri Url { get; set; }

    public TimeSpan RequestDuration { get; set; }

}

}
```

ConsulLoader.cs (Monitoring.ETL.Domain\Consul\ConsulLoader.cs):

```
using System;
```

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.Consul
```

```
{  
  
    public abstract class ConsulLoader  
  
    {  
  
        private readonly IWarehouseServerProvider _warehouseServerProvider;
```

```
  
        protected ConsulLoader()  
  
            : this(new WarehouseServerProvider())  
  
        {  
  
        }  
    }
```

```
  
    internal ConsulLoader(IWarehouseServerProvider warehouseServerProvider)  
  
    {  
  
        _warehouseServerProvider = warehouseServerProvider ?? throw new  
ArgumentNullException(nameof(warehouseServerProvider));  
  
    }
```

```
  
    protected string GetWarehouseConnectionString()  
  
    {  
  
        var warehouseServerName = _warehouseServerProvider.GetWarehouseServerName();  
  
  
        return $"data source={warehouseServerName};initial catalog=Performance;integrated security=True";  
  
    }  
  
    }  
  
}
```

KeyValueExtractionDelayFactory.cs (Monitoring.ETL.Domain\Consul\KeyValueExtractionDelayFactory.cs):

using System;

using Monitoring.ETL.Domain.Consul.Model;

namespace Monitoring.ETL.Domain.Consul

{

public sealed class KeyValueExtractionDelayFactory : PeriodicExtractionDelayFactory<KeyValueModel>

{

protected override TimeSpan DelayTimeSpan => new TimeSpan(0, 2, 0);

}

}

KeyValueExtractor.cs (Monitoring.ETL.Domain\Consul\KeyValueExtractor.cs):

using System;

using System.Collections.Generic;

using ETL.Process;

using Monitoring.ETL.Domain.Consul.Model;

namespace Monitoring.ETL.Domain.Consul

{

public sealed class KeyValueExtractor : ConsulExtractor<KeyValueModel>, IExtractor<KeyValueModel>

{

protected override IEnumerable<KeyValueModel> GetModels(ConsulHttpClient consulHttpClient, out
IList<Exception> exceptions)

{

return consulHttpClient.GetKeyValuePairs(out exceptions);

```
}  
  
}  
  
}
```

KeyValueLoader.cs (Monitoring.ETL.Domain\Consul\KeyValueLoader.cs):

```
using System;  
  
using System.Collections.Generic;  
  
using System.Data;  
  
using System.Data.SqlClient;  
  
using System.Threading;  
  
using System.Threading.Tasks;  
  
using ETL.Process;  
  
using Monitoring.ETL.Domain.Consul.Model;  
  
using Monitoring.ETL.Process;  
  
using Newtonsoft.Json;  
  
  
namespace Monitoring.ETL.Domain.Consul  
{  
  
    public sealed class KeyValueLoader : ConsulLoader, ILoader<KeyValueModel>  
    {  
  
        public async Task<bool> LoadAsync(IEnumerable<KeyValueModel> transformed, CancellationToken  
cancellationToken, IEtlProcessLogger logger)  
  
        {  
  
            var jsonString = JsonConvert.SerializeObject(transformed);  
  
  
            using (var con = new SqlConnection(GetWarehouseConnectionString()))
```

```
{

con.Open();

using (var cmd = new SqlCommand("Meta_Warehouse.dbo.Dim_Consul_Key_Value_ETL", con))

{

    cmd.Parameters.Add(

        new SqlParameter

        {

            ParameterName = "@Dim_Consul_Key_Value_JSON",

            SqlDbType = SqlDbType.VarChar,

            Size = jsonString.Length,

            Value = jsonString,

        });

    cmd.Parameters.Add(

        new SqlParameter

        {

            ParameterName = "@Event_Date",

            SqlDbType = SqlDbType.DateTime,

            Size = 8,

            Value = DateTime.Now

        });

    cmd.CommandType = CommandType.StoredProcedure;

    cmd.ExecuteNonQuery();

}
```

```

    }

}

return true;

}

}

}

```

DataCenterModel.cs (Monitoring.ETL.Domain\Consul\Model\DataCenterModel.cs):

```

namespace Monitoring.ETL.Domain.Consul.Model

{

    public sealed class DataCenterModel

    {

        public string Name { get; set; }

        public string Environment { get; set; }


        public override string ToString()

        {

            return $"DataCenterModel: {{ Env: \"{Environment}\" DC: \"{Name}\" }}";

        }

    }

}

```

HealthCheckModel.cs (Monitoring.ETL.Domain\Consul\Model\HealthCheckModel.cs):

```

using System.Collections.Generic;

```



```
namespace Monitoring.ETL.Domain.Consul.Model
```

```
{  
  
    public sealed class HealthCheckModel  
  
    {  
  
        public string Environment { get; set; }  
  
        public string DataCenter { get; set; }  
  
        public string Node { get; set; }  
  
        public string Name { get; set; }  
  
        public string CheckId { get; set; }  
  
        public string ServiceId { get; set; }  
  
        public string ServiceName { get; set; }  
  
        public List<string> ServiceTags { get; set; }  
  
        public override string ToString()  
  
        {  
  
            var formattedTags = ServiceTags != null && ServiceTags.Count > 0 ? string.Join(", ", ServiceTags) : string.Empty;  
  
            return $"HealthCheckModel {{ Env: \"{Environment}\" DC: \"{DataCenter}\" Name: \"{Name}\" ServiceName:  
\"{ServiceName}\" ServiceId: \"{ServiceId}\" Node: \"{Node}\" Tags: [{formattedTags}] }}";  
  
        }  
  
    }  
  
}
```

HostNamePortModel.cs (Monitoring.ETL.Domain\Consul\Model\HostNamePortModel.cs):

```
namespace Monitoring.ETL.Domain.Consul.Model  
  
{  
  
    public sealed class HostNamePortModel
```

```

{

    public string HostName { get; set; }

    public int? Port { get; set; }


    public override string ToString()

    {

        return $"HostNamePortModel {{ HostName: \"{HostName}\" Port: \"{Port}\" }}";

    }

}

}

```

KeyValueModel.cs (Monitoring.ETL.Domain\Consul\Model\KeyValueModel.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Consul.Model
```

```

{

    public sealed class KeyValueModel : IExtractModel, ILoadModel

    {

        public string Environment { get; set; }

        public string DataCenter { get; set; }

        public string Key { get; set; }

        public string Value { get; set; }


        public override string ToString()

        {

            return $"KeyValueModel: {{ Env: \"{Environment}\" DC: \"{DataCenter}\" Key: \"{Key}\" Value: \"{Value}\" }}";


```

```
}  
  
}  
  
}
```

NodeModel.cs (Monitoring.ETL.Domain\Consul\Model\NodeModel.cs):

```
using System.Collections.Generic;
```

```
using Newtonsoft.Json;
```

```
namespace Monitoring.ETL.Domain.Consul.Model
```

```
{  
  
    public sealed class NodeModel  
  
    {  
  
        public string Id { get; set; }  
  
        [JsonProperty("Node")]  
  
        public string Name { get; set; }  
  
        public string Environment { get; set; }  
  
        public string DataCenter { get; set; }  
  
        public string Address { get; set; }  
  
        public Dictionary<string, HostNamePortModel> ServiceAddressHostNames { get; set; }  
  
  
        public override string ToString()  
  
        {  
  
            return $"NodeModel: {{ Env: \"{Environment}\" DC: \"{DataCenter}\" Name: \"{Name}\" Address: \"{Address}\" }}";  
  
        }  
  
    }  
  
}
```

ServiceModel.cs (Monitoring.ETL.Domain\Consul\Model\ServiceModel.cs):

```
using Newtonsoft.Json;
```

```
namespace Monitoring.ETL.Domain.Consul.Model
```

```
{
```

```
    public sealed class ServiceModel
```

```
    {
```

```
        public string Id { get; set; }
```

```
        [JsonProperty("Service")]
```

```
        public string Name { get; set; }
```

```
        public string Environment { get; set; }
```

```
        public string DataCenter { get; set; }
```

```
        public string Address { get; set; }
```

```
        public int? Port { get; set; }
```

```
        public override string ToString()
```

```
        {
```

```
            return $"ServiceModel: {{ Env: \"{Environment}\" DC: \"{DataCenter}\" Name: \"{Name}\" Address: \"{Address}\" }}";
```

```
        }
```

```
    }
```

```
}
```

ServiceNodeMappingModel.cs (Monitoring.ETL.Domain\Consul\Model\ServiceNodeMappingModel.cs):

```
using System.Collections.Generic;
```

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Consul.Model
```

```
{

    public sealed class ServiceNodeMappingModel : IExtractModel, ILoadModel

    {

        public string ServiceName { get; set; }

        public string NodeName { get; set; }

        public int? Port { get; set; }

        public string Environment { get; set; }

        public string DataCenter { get; set; }

        public List<string> ServiceTags { get; set; }


        public override string ToString()

        {

            var formattedTags = ServiceTags != null && ServiceTags.Count > 0 ? string.Join(",", ServiceTags) : string.Empty;

            return $"ServiceNodeMappingModel: {{ Env: \"{Environment}\" DC: \"{DataCenter}\" ServiceName: \"{ServiceName}\"

NodeName: \"{NodeName}\" Port: \"{Port}\" Tags: [{formattedTags}] }}";

        }

    }

}
```

```
ServiceNodeMappingDelayFactory.cs (Monitoring.ETL.Domain\Consul\ServiceNodeMappingDelayFactory.cs):
```

```
using System;
```

```
using Monitoring.ETL.Domain.Consul.Model;
```

```
namespace Monitoring.ETL.Domain.Consul
```

```

{

    public sealed class ServiceNodeMappingDelayFactory : PeriodicExtractionDelayFactory<ServiceNodeMappingModel>

    {

        protected override TimeSpan DelayTimeSpan => new TimeSpan(0, 2, 0);

    }

}

```

ServiceNodeMappingExtractor.cs (Monitoring.ETL.Domain\Consul\ServiceNodeMappingExtractor.cs):

```

using System;

using System.Collections.Generic;

using ETL.Process;

using Monitoring.ETL.Domain.Consul.Model;

namespace Monitoring.ETL.Domain.Consul

{

    public sealed class ServiceNodeMappingExtractor : ConsulExtractor<ServiceNodeMappingModel>,

    IExtractor<ServiceNodeMappingModel>

    {

        protected override IEnumerable<ServiceNodeMappingModel> GetModels(ConsulHttpClient consulHttpClient,

out IList<Exception> exceptions)

        {

            return consulHttpClient.GetServiceNodeMappings(out exceptions);

        }

    }

}

```

ServiceNodeMappingLoader.cs (Monitoring.ETL.Domain\Consul\ServiceNodeMappingLoader.cs):

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SqlClient;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Domain.Consul.Model;

using Monitoring.ETL.Process;

using Newtonsoft.Json;

namespace Monitoring.ETL.Domain.Consul

{

public sealed class ServiceNodeMappingLoader : ConsulLoader, ILoader<ServiceNodeMappingModel>

{

public async Task<bool> LoadAsync(IEnumerable<ServiceNodeMappingModel> transformed, CancellationToken
cancellationToken, IEtlProcessLogger logger)

{

var jsonString = JsonConvert.SerializeObject(transformed);

using (var con = new SqlConnection(GetWarehouseConnectionString()))

{

con.Open();

using (var cmd = new SqlCommand("Meta_Warehouse.dbo.Dim_Consul_Service_Node_Mapping_ETL", con))

```
{

cmd.Parameters.Add(

    new SqlParameter

    {

        ParameterName = "@Dim_Consul_Service_Node_Mapping_JSON",

        SqlDbType = SqlDbType.VarChar,

        Size = jsonString.Length,

        Value = jsonString,

    });


cmd.Parameters.Add(

    new SqlParameter

    {

        ParameterName = "@Event_Date",

        SqlDbType = SqlDbType.DateTime,

        Size = 8,

        Value = DateTime.Now

    });


cmd.CommandType = CommandType.StoredProcedure;


cmd.ExecuteNonQuery();

}

}

return true;
```



```
}  
  
}  
  
}
```

ConsulConnectionRepository.cs (Monitoring.ETL.Domain\ConsulConnectionRepository.cs):

```
using System;  
  
using System.Collections.Generic;  
  
using System.Data.SqlClient;  
  
using System.Linq;  
  
using System.Xml.Linq;  
  
  
namespace Monitoring.ETL.Domain  
{  
  
    internal class ConsulConnectionFactory  
  
    {  
  
        private readonly Dictionary<string, ConsulConfiguration> _configurations;  
  
  
        /// <summary>  
        /// Populate the list of configurations from the infrastructure.xml file  
        /// </summary>  
  
        public ConsulConnectionFactory()  
  
        {  
  
            var infraXml = @"c:\infrastructure.xml";  
  
            var xmlDoc = XDocument.Load(infraXml);  
  
  
            var services = xmlDoc.Root
```

```
? .Element("services")

?.Element("databases")

?.Elements("service").ToList();
```

```
if (services != null && services.Any())
```

```
{
```

```
    _configurations = services
```

```
        .Select(e => new ConsulConfiguration
```

```
        {
```

```
            Id = (string)e.Attribute("id"),
```

```
            Host = e.Attribute("host")?.Value,
```

```
            Port = e.Attribute("port")?.Value,
```

```
            Username = e.Attribute("username")?.Value,
```

```
            Password = e.Attribute("password")?.Value
```

```
        })
```

```
        .ToDictionary(c => c.Id);
```

```
}
```

```
else
```

```
{
```

```
    throw new ArgumentNullException(nameof(services),
```

```
        $"{infraXml} is either not found, or a valid service is not found (root->services->databases->service);
```

```
}
```

```
}
```

```
/// <summary>
```

```
/// Create a database connection string based on a consul ID
```

```
/// </summary>
```

```
/// <param name="id">The consule id</param>
```

```
/// <param name="database">The database</param>
```

```
/// <returns></returns>
```

```
public SqlConnection Create(string id, string database)
```

```
{
```

```
    id = id.ToLower();
```

```
    if (!_configurations.ContainsKey(id))
```

```
        throw new Exception($"Consul id: {id} not found");
```

```
    var configuration = _configurations[id];
```

```
    string connectionString = "data source={0},{1}";
```

```
    if (string.IsNullOrEmpty(database) == false)
```

```
        connectionString += "initial catalog={4}";
```

```
    if (configuration.UserName != null && configuration.Password != null)
```

```
        connectionString += "user={2};password={3}";
```

```
    else
```

```
        connectionString += "Integrated Security=true";
```

```
    connectionString = string.Format(
```

```
        connectionString,
```

```
        configuration.Host,
```

```

        configuration.Port,

        configuration.UserName,

        configuration.Password,

        database);

    return new SqlConnection(connectionString);
}

private class ConsulConfiguration
{
    public string Id { get; set; }

    public string Host { get; set; }

    public string Port { get; set; }

    public string UserName { get; set; }

    public string Password { get; set; }
}
}
}

```

CustomerDelayFactory.cs (Monitoring.ETL.Domain\Customer\CustomerDelayFactory.cs):

```

using System;

namespace Monitoring.ETL.Domain.Customer
{
    public class CustomerDelayFactory : ThresholdExtractionDelayFactory<RabbitMQ.Model.Customer>
    {

```

```
protected override int MaxThresholdForDelay => int.MaxValue;
```

```
protected override TimeSpan DelayTimeSpan => TimeSpan.FromMinutes(1);
```

```
}
```

```
}
```

CustomerShard.cs (Monitoring.ETL.Domain\Customer\CustomerShard.cs):

```
namespace Monitoring.ETL.Domain.Customer
```

```
{
```

```
    /// <summary>
```

```
    /// PCN and shard info from http://dv-nscale-p01/v1/info
```

```
    /// </summary>
```

```
    public class CustomerShard
```

```
    {
```

```
        /// <summary>
```

```
        /// The PCN
```

```
        /// </summary>
```

```
        public int Id { get; set; }
```

```
        /// <summary>
```

```
        /// The Shard
```

```
        /// </summary>
```

```
        public string Shard { get; set; }
```

```
    }
```

```
}
```

CustomerTransformer.cs (Monitoring.ETL.Domain\Customer\CustomerTransformer.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Customer
```

```
{
```

```
    public class CustomerTransformer : ITransformer<RabbitMQ.Model.Customer, Warehouse.Model.Customer_w>
```

```
    {
```

```
        public async Task<IEnumerable<Warehouse.Model.Customer_w>>
```

```
TransformAsync(IEnumerable<RabbitMQ.Model.Customer> extracted, CancellationToken cancellationToken,
```

```
IEtlProcessLogger logger)
```

```
    {
```

```
        await Task.Yield();
```

```
        return extracted.Select(pc =>
```

```
            new Warehouse.Model.Customer_w
```

```
            {
```

```
                JSON_Message = pc.JsonMessage
```

```
            });
```

```
    }
```

```
}
```

```
}
```

PlexusCustomer.cs (Monitoring.ETL.Domain\Customer\PlexusCustomer.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Customer
```

```
{  
  
    public class PlexusCustomer : IExtractModel, SqlJson.IJsonExtractModel  
  
    {  
  
        public string JsonMessage { get; set; }  
  
    }  
}
```

PlexusCustomerDelayFactory.cs (Monitoring.ETL.Domain\Customer\PlexusCustomerDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Customer
```

```
{  
  
    public class PlexusCustomerDelayFactory : TimeOfDayExtractionDelayFactory<PlexusCustomer>  
  
    {  
  
        protected override TimeSpan TimeOfDay => new TimeSpan(3, 30, 00);  
  
    }  
}
```

PlexusCustomerExtractor.cs (Monitoring.ETL.Domain\Customer\PlexusCustomerExtractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.HelperDataset.CustomerUxClassic;
```

```
using Monitoring.ETL.Domain.HelperDataset.Salesforce;
```

```
using Monitoring.ETL.Domain.HelperDataset.Shard;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Customer
```

```
{
```

```
    public class PlexusCustomerExtractor : IExtractor<PlexusCustomer>
```

```
    {
```

```
        private readonly SqlJson.Repository<PlexusCustomer> _repository;
```

```
        public PlexusCustomerExtractor()
```

```
        {
```

```
            _repository = new SqlJson.Repository<PlexusCustomer>(
```

```
                new List<string> { EtlSettings.ServerId },
```

```
                "Plexus_Control",
```

```
                @"SELECT
```

```
                PC.Plexus_Customer_No AS PCN,
```

```
                PC.Plexus_Customer_Code,
```

```
                PC.[Name] AS Customer_Name,
```



```

CG.[Name] AS Customer_Group_Name,

CG.Customer_Group_No,

CG.Primary_Plexus_Customer_No AS Customer_Group_Primary_PCN,

PC.Country,

TZ.[Description] AS Timezone,

TZ.Timezone_Offset,

PC.Plexus_SQL_Server_Group_Key,

PC.Add_Date,

C.Customer_Status,

C.Customer_Type,

CS.Active,

CS.Accounting_Active,

PC.Restrict_Access_To_US_Citizens AS ITAR_Customer,

CASE

    WHEN C.Customer_Type IN ('Education', 'Automated Testing', 'Demo', 'Plex Employee', 'QA Testing',
'Template') THEN 1

    WHEN C.Customer_Type = 'Partner' THEN 0

    WHEN C.Customer_Status = 'Partner-Active' THEN 0

    WHEN C.Name LIKE '%test%' THEN 1

    WHEN C.Name LIKE '%sandbox%' THEN 1

    WHEN C.Name LIKE '%Plex%' THEN 1

    WHEN C.Name LIKE '%PCN%' THEN 1

    WHEN C.Name LIKE '%template%' THEN 1

    WHEN CGM.Customer_Group_No IN

    (

        13, --Plex

```

21, --Edge

180, --POL Template Group

256, --Customer Care Testing

467, --TechDoc

607, --Global Mfg 2

780, --Plex Customer Care

797, --Control+M Solutions COLO

823, --IAM

853 --Pro Serv Testing

)

THEN 1

ELSE 0

END AS Internal,

CASE

WHEN C.Customer_Type IN ('Education', 'Automated Testing', 'Demo', 'Plex Employee', 'QA Testing',
'Template') THEN 0

WHEN C.Customer_Type = 'Partner' THEN 1

WHEN C.Customer_Status = 'Partner-Active' THEN 1

WHEN EXISTS

(

SELECT

*

FROM Plexus_Control.dbo.Plexus_Customer_Partner AS PCP

WHERE PCP.Partner_PCN = PC.Plexus_Customer_No

)

THEN 1

```

ELSE 0

END AS [Partner]

FROM Plexus_Control.dbo.Plexus_Customer AS PC

JOIN Common.dbo.Customer AS C

ON C.Plexus_Customer_No = 1

AND C.Customer_No = PC.Plexus_Customer_No

JOIN Common.dbo.Customer_Status AS CS

ON CS.Plexus_Customer_No = C.Plexus_Customer_No

AND CS.Customer_Status = C.Customer_Status

JOIN Plexus_Control.dbo.Logical_Timezone AS TZ

ON TZ.Timezone_Key = PC.Timezone_Key

LEFT OUTER JOIN Plexus_Control.dbo.Customer_Group_Member AS CGM

ON CGM.Plexus_Customer_No = PC.Plexus_Customer_No

LEFT OUTER JOIN Plexus_Control.dbo.Customer_Group AS CG

ON CG.Customer_Group_No = CGM.Customer_Group_No

ORDER BY PC.[Name]

FOR JSON PATH;");

}

```

```

public async Task<ResultSet<PlexusCustomer>> ExtractAllSinceLastExtractAsync(CancellationToken
cancellationTokens, IETIProcessLogger logger)

{

    var result = _repository.ExtractAllSinceLastExtractAsync(cancellationTokens, logger).Result;

    logger.Information("Extracting Shard Information");

    var shardClient = new ShardClient(logger);

```

```
result = shardClient.AppendDataset(result);
```

```
logger.Information("Extracting Salesforce Information");
```

```
var salesforce = new SalesforceClient(logger);
```

```
result = salesforce.AppendDataset(result);
```

```
logger.Information("Extracting UX/Classic Information");
```

```
var uxClassic = new CustomerUxClassicClient(logger);
```

```
result = uxClassic.AppendDataset(result);
```

```
return result;
```

```
}
```

```
public Task<IEnumerable<PlexusCustomer>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,  
Cancellation token cancellation token, IETLProcessLogger logger)
```

```
{
```

```
    throw new NotImplementedException();
```

```
}
```

```
}
```

```
}
```

PlexusCustomerTransformer.cs (Monitoring.ETL.Domain\Customer\PlexusCustomerTransformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Customer
```

```
{
```

```
    public class Transformer : ITransformer<PlexusCustomer, RabbitMQ.Model.Customer>
```

```
    {
```

```
        public async Task<IEnumerable<RabbitMQ.Model.Customer>> TransformAsync(IEnumerable<PlexusCustomer>
```

```
extracted, CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
        {
```

```
            await Task.Yield();
```

```
            return extracted.Select(pc =>
```

```
                new RabbitMQ.Model.Customer
```

```
                {
```

```
                    JsonMessage = pc.JsonMessage
```

```
                });
```

```
        }
```

```
    }
```

```
}
```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\Customer\RabbitMQExtractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Customer
```

```
{  
  
    public sealed class RabbitMQExtractor : IExtractor<RabbitMQ.Model.Customer>
```

```
{  
  
    private RabbitMQ.Customer.Consumer _consumer;
```

```
  
    public async Task<ResultSet<RabbitMQ.Model.Customer>> ExtractAllSinceLastExtractAsync(CancellationTok  
cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    _consumer = _consumer ?? new RabbitMQ.Customer.Consumer(logger);  
  
    return await _consumer.ExtractAsync();  
  
}
```

```
  
    public Task<IEnumerable<RabbitMQ.Model.Customer>> ExtractBetweenAsync(DateTime startDate, DateTime  
endDate, CancellationTok  
cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    throw new NotImplementedException();  
  
}
```

```
}
```

```
}
```

RabbitMQLoader.cs (Monitoring.ETL.Domain\Customer\RabbitMQLoader.cs):

```
namespace Monitoring.ETL.Domain.Customer
```

```

{

    public class RabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.Customer>

    {

        public RabbitMQLoader()

            : base(new RabbitMQ.Customer.Producer())

        {

        }

    }

}

```

WarehouseLoader.cs (Monitoring.ETL.Domain\Customer\WarehouseLoader.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.Customer
```

```

{

    public class WarehouseLoader : Loader<Warehouse.Model.Customer_w>

    {

        public WarehouseLoader() : base("Dim_Customer_Add")

        {

        }

    }

}

```

AzureDevOpsBuild.cs (Monitoring.ETL.Domain\Deployment\AzureDevOps\AzureDevOpsBuild.cs):

```
using Monitoring.ETL.Domain.AzureQueue;
```

```
namespace Monitoring.ETL.Domain.Deployment.AzureDevOps
```

```
{  
  
    public class AzureDevOpsBuild : IAzureQueueMessage  
  
    {  
  
        public string JsonMessage { get; set; }  
  
    }  
  
}
```

AzureDevOpsRelease.cs (Monitoring.ETL.Domain\Deployment\AzureDevOps\AzureDevOpsRelease.cs):

```
using Monitoring.ETL.Domain.AzureQueue;
```

```
namespace Monitoring.ETL.Domain.Deployment.AzureDevOps
```

```
{  
  
    public class AzureDevOpsRelease : IAzureQueueMessage  
  
    {  
  
        public string JsonMessage { get; set; }  
  
    }  
  
}
```

BuildDelayFactory.cs (Monitoring.ETL.Domain\Deployment\AzureDevOps\BuildDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Deployment.AzureDevOps
```

```
{  
  
    public class BuildDelayFactory : ThresholdExtractionDelayFactory<RabbitMQ.Model.Build>  
  
    {
```



```
protected override int MaxThresholdForDelay => 100;
```

```
protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);
```

```
}
```

```
}
```

BuildExtractionDelayFactory.cs (Monitoring.ETL.Domain\Deployment\AzureDevOps\BuildExtractionDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Deployment.AzureDevOps
```

```
{
```

```
public class BuildExtractionDelayFactory : ThresholdExtractionDelayFactory<AzureDevOpsBuild>
```

```
{
```

```
protected override int MaxThresholdForDelay => 1;
```

```
protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);
```

```
}
```

```
}
```

BuildExtractor.cs (Monitoring.ETL.Domain\Deployment\AzureDevOps\BuildExtractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Deployment.AzureDevOps
```

```
{
```

```
    public class BuildExtractor : IExtractor<AzureDevOpsBuild>
```

```
    {
```

```
        private AzureQueue.Repository<AzureDevOpsBuild> _repository;
```

```
        public BuildExtractor()
```

```
        {
```

```
            _repository = new AzureQueue.Repository<AzureDevOpsBuild>("azdo-builds");
```

```
        }
```

```
        public async Task<ResultSet<AzureDevOpsBuild>> ExtractAllSinceLastExtractAsync(CancellationToken
```

```
cancellationTokens, IEtlProcessLogger logger)
```

```
        {
```

```
            var resultSet = new ResultSet<AzureDevOpsBuild>();
```

```
            foreach (var build in await _repository.Get(100, cancellationTokens))
```

```
            {
```

```
                resultSet.Results.Add(build);
```

```
            }
```

```
            return resultSet;
```

```
        }
```

```
        public Task<IEnumerable<AzureDevOpsBuild>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
```

```
CancellationToken cancellationTokens, IEtlProcessLogger logger)
```

```

{
    throw new NotImplementedException();
}
}
}

```

BuildRabbitMQExtractor.cs (Monitoring.ETL.Domain\Deployment\AzureDevOps\BuildRabbitMQExtractor.cs):

```

using System;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.Deployment.AzureDevOps
{
    public sealed class BuildRabbitMQExtractor : IExtractor<RabbitMQ.Model.Build>
    {
        private RabbitMQ.Build.Consumer _consumer;

        public async Task<ResultSet<RabbitMQ.Model.Build>> ExtractAllSinceLastExtractAsync(CancellationTok
cancellationToken, IEtlProcessLogger logger)
        {
            _consumer = _consumer ?? new RabbitMQ.Build.Consumer(logger);

            return await _consumer.ExtractAsync();
        }
    }
}

```

```
public Task<IEnumerable<RabbitMQ.Model.Build>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
CancellationTokens cancellationTokens, IEtlProcessLogger logger)
{
    throw new NotImplementedException();
}
}
```

BuildRabbitMQLoader.cs (Monitoring.ETL.Domain\Deployment\AzureDevOps\BuildRabbitMQLoader.cs):

```
namespace Monitoring.ETL.Domain.Deployment.AzureDevOps.Release
{
    public class BuildRabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.Build>
    {
        public BuildRabbitMQLoader()
            : base(new RabbitMQ.Build.Producer())
        {
        }
    }
}
```

BuildRabbitMQTransformer.cs (Monitoring.ETL.Domain\Deployment\AzureDevOps\BuildRabbitMQTransformer.cs):

```
using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Deployment.AzureDevOps
```

```
{
```

```
    public class BuildRabbitMQTransformer : ITransformer<AzureDevOpsBuild, RabbitMQ.Model.Build>
```

```
    {
```

```
        public async Task<IEnumerable<RabbitMQ.Model.Build>> TransformAsync(IEnumerable<AzureDevOpsBuild>
```

```
extracted, CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
        {
```

```
            return extracted.Select(e => new RabbitMQ.Model.Build { JsonMessage = e.JsonMessage });
```

```
        }
```

```
    }
```

```
}
```

```
BuildRabbitMQWarehouseTransformer.cs
```

```
(Monitoring.ETL.Domain\Deployment\AzureDevOps\BuildRabbitMQWarehouseTransformer.cs):
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.Warehouse.Model;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Deployment.AzureDevOps
```

```

{

    public class BuildRabbitMQWarehouseTransformer : ITransformer<RabbitMQ.Model.Build, Build_w>

    {

        public async Task<IEnumerable<Build_w>> TransformAsync(IEnumerable<RabbitMQ.Model.Build> extracted,
CancellationTokentoken, IEtlProcessLogger logger)

        {

            await Task.Yield();

            return extracted.Select(r => new Build_w

            {

                JSON_Message = r.JsonMessage

            });

        }

    }

}

```

BuildWarehouseLoader.cs (Monitoring.ETL.Domain\Deployment\AzureDevOps\BuildWarehouseLoader.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.Deployment.AzureDevOps
```

```

{

    public class BuildWarehouseLoader : Loader<Warehouse.Model.Build_w>

    {

        public BuildWarehouseLoader()

        : base("Fact_Build_Add")

        {

        }

    }

}

```

```
}
```

```
}
```

ReleaseDelayFactory.cs (Monitoring.ETL.Domain\Deployment\AzureDevOps\ReleaseDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Deployment.AzureDevOps
```

```
{
```

```
    public class ReleaseDelayFactory : ThresholdExtractionDelayFactory<RabbitMQ.Model.Release>
```

```
    {
```

```
        protected override int MaxThresholdForDelay => 100;
```

```
        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);
```

```
    }
```

```
}
```

ReleaseExtractionDelayFactory.cs

(Monitoring.ETL.Domain\Deployment\AzureDevOps\ReleaseExtractionDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Deployment.AzureDevOps
```

```
{
```

```
    public class ReleaseExtractionDelayFactory : ThresholdExtractionDelayFactory<AzureDevOpsRelease>
```

```
    {
```

```
        protected override int MaxThresholdForDelay => 1;
```

```

        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);
    }
}

ReleaseExtractor.cs (Monitoring.ETL.Domain\Deployment\AzureDevOps\ReleaseExtractor.cs):

using System;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.Deployment.AzureDevOps
{
    public class ReleaseExtractor : IExtractor<AzureDevOpsRelease>
    {
        private AzureQueue.Repository<AzureDevOpsRelease> _repository;

        public ReleaseExtractor()
        {
            _repository = new AzureQueue.Repository<AzureDevOpsRelease>("azdo-releases");
        }

        public async Task<ResultSet<AzureDevOpsRelease>> ExtractAllSinceLastExtractAsync(CancellationTok
cancellationToken, IEtlProcessLogger logger)
        {

```



```

var resultSet = new ResultSet<AzureDevOpsRelease>();

foreach (var release in await _repository.Get(20, cancellationToken))

{

    resultSet.Results.Add(release);

}


return resultSet;

}


public Task<IEnumerable<AzureDevOpsRelease>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
Cancellation token cancellationToken, IEtlProcessLogger logger)

{

    throw new NotImplementedException();

}

}

}

```

ReleaseRabbitMQExtractor.cs (Monitoring.ETL.Domain\Deployment\AzureDevOps\ReleaseRabbitMQExtractor.cs):

```

using System;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

```

```

namespace Monitoring.ETL.Domain.Deployment.AzureDevOps

```

```

{

public sealed class ReleaseRabbitMQExtractor : IExtractor<RabbitMQ.Model.Release>

{

private RabbitMQ.Release.Consumer _consumer;


public async Task<ResultSet<RabbitMQ.Model.Release>> ExtractAllSinceLastExtractAsync(CancellationTok
cancellationToken, IEtlProcessLogger logger)

{

_consumer = _consumer ?? new RabbitMQ.Release.Consumer(logger);

return await _consumer.ExtractAsync();

}


public Task<IEnumerable<RabbitMQ.Model.Release>> ExtractBetweenAsync(DateTime startDate, DateTime
endDate, CancellationTok cancellationToken, IEtlProcessLogger logger)

{

throw new NotImplementedException();

}

}

}

```

ReleaseRabbitMQLoader.cs (Monitoring.ETL.Domain\Deployment\AzureDevOps\ReleaseRabbitMQLoader.cs):

```

namespace Monitoring.ETL.Domain.Deployment.AzureDevOps.Release

{

public class ReleaseRabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.Release>

{

public ReleaseRabbitMQLoader()

```

```

        : base(new RabbitMQ.Release.Producer())

    {

    }

}

}

```

ReleaseRabbitMQTransformer.cs

(Monitoring.ETL.Domain\Deployment\AzureDevOps\ReleaseRabbitMQTransformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Deployment.AzureDevOps
```

```

{

    public class ReleaseRabbitMQTransformer : ITransformer<AzureDevOpsRelease, RabbitMQ.Model.Release>

    {

        public async Task<IEnumerable<RabbitMQ.Model.Release>> TransformAsync(IEnumerable<AzureDevOpsRelease>
extracted, CancellationToken cancellationToken, IEtlProcessLogger logger)

        {

            return extracted.Select(e => new RabbitMQ.Model.Release { JsonMessage = e.JsonMessage });

        }

    }

}

```

ReleaseRabbitMQWarehouseTransformer.cs

(Monitoring.ETL.Domain\Deployment\AzureDevOps\ReleaseRabbitMQWarehouseTransformer.cs):

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Domain.Warehouse.Model;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.Deployment.AzureDevOps

{

public class ReleaseRabbitMQWarehouseTransformer : ITransformer<RabbitMQ.Model.Release, Release_w>

{

public async Task<IEnumerable<Release_w>> TransformAsync(IEnumerable<RabbitMQ.Model.Release> extracted,

CancellationToken cancellationToken, IEtlProcessLogger logger)

{

int i = 1;

await Task.Yield();

return extracted.Select(r => new Release_w

{

JSON_Message = r.JsonMessage,

Load_Event_Number = i++

});

}

}

```
}
```

ReleaseWarehouseLoader.cs (Monitoring.ETL.Domain\Deployment\AzureDevOps\ReleaseWarehouseLoader.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.Deployment
```

```
{
```

```
    public class ReleaseWarehouseLoader : Loader<Warehouse.Model.Release_w>
```

```
    {
```

```
        public ReleaseWarehouseLoader()
```

```
            : base("Fact_Deployment_Add")
```

```
        {
```

```
        }
```

```
    }
```

```
}
```

ClassicDeployment.cs (Monitoring.ETL.Domain\Deployment\Classic\ClassicDeployment.cs):

```
using System;
```

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Deployment.Classic
```

```
{
```

```
    public class ClassicDeployment : IExtractModel
```

```
    {
```

```
        public int Deployment_Key { get; set; }
```

```
        public DateTime Commit_Date { get; internal set; }
```

```

public DateTime Deployment_Date { get; internal set; }

public string Deployed_To_DB_Servers { get; internal set; }

public int Directed_Deployment_Plexus_Server_Group_Key { get; internal set; }

public string Deployed_To_Webservers { get; internal set; }

public int Owner { get; internal set; }

public int Deployed_By { get; internal set; }

//public int Classic_Application_Key { get; internal set; }

//public string Filename { get; internal set; }

//public int SQL_Object_Key { get; internal set; }

//public string Script { get; internal set; }

public string Cart_Name { get; internal set; }

public bool Rollback_Deployment { get; internal set; }

public bool Duplicate_Deployment { get; internal set; }

public bool Multi_Line_Package_Deployment { get; internal set; }

public int Copy_Of_Deployment_Key { get; internal set; }

public int First_Deployment_Key { get; internal set; }

public string Project_Name { get; internal set; }

public string Database_Name { get; internal set; }

public string Environment { get; internal set; }

public string Service_Tickets { get; internal set; }

}

}

```

DelayFactory.cs (Monitoring.ETL.Domain\Deployment\Classic\DelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Deployment.Classic
```

```
{  
  
    public class DelayFactory : ThresholdExtractionDelayFactory<ClassicDeployment>  
  
    {  
  
        protected override int MaxThresholdForDelay => 100;  
  
  
        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);  
  
    }  
}
```

```
Extractor.cs (Monitoring.ETL.Domain\Deployment\Classic\Extractor.cs):
```

```
using System;  
  
using System.Collections.Generic;  
  
using System.Linq;  
  
using System.Threading;  
  
using System.Threading.Tasks;  
  
  
using ETL.Process;  
  
  
using Monitoring.ETL.Domain.ElasticSearch;  
  
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Deployment.Classic
```

```
{  
  
    public class Extractor : IExtractor<ClassicDeployment>  
  
    {
```

```
private readonly LoadStateRepository<ClassicDeployment> _loadStateRepository;
```

```
private readonly Repository _repository;
```

```
private DateTime _lastDate;
```

```
public Extractor()
```

```
{
```

```
    _loadStateRepository = new LoadStateRepository<ClassicDeployment>();
```

```
    _repository = new Repository();
```

```
}
```

```
public async Task<ResultSet<ClassicDeployment>> ExtractAllSinceLastExtractAsync(CancellationTokencancellationTokentoken, IETIProcessLogger logger)
```

```
{
```

```
    var resultSet = new ResultSet<ClassicDeployment>();
```

```
    try
```

```
    {
```

```
        if (_lastDate == default)
```

```
        {
```

```
            _lastDate = _loadStateRepository.GetDate();
```

```
        }
```

```
        var deployments = await _repository.GetAllAfterDate(_lastDate, cancellationToken);
```

```
        foreach (var deployment in deployments)
```

```
        {
```

```
            resultSet.Results.Add(deployment);
```



```
}
```

```
var max = deployments
```

```
.Select(d => d.Deployment_Date)
```

```
.OrderByDescending(date => date)
```

```
.FirstOrDefault();
```

```
if (max > _lastDate)
```

```
{
```

```
    _lastDate = max;
```

```
    await _loadStateRepository.SetDate(_lastDate);
```

```
}
```

```
}
```

```
catch (Exception ex)
```

```
{
```

```
    resultSet.Exceptions.Add(ex);
```

```
}
```

```
return resultSet;
```

```
}
```

```
public Task<IEnumerable<ClassicDeployment>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,  
Cancellation token cancellation token, IEtlProcessLogger logger)
```

```
{
```

```
    throw new NotImplementedException();
```

```
}
```

```
}
```

```
}
```

RabbitMQTransformer.cs (Monitoring.ETL.Domain\Deployment\Classic\RabbitMQTransformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Deployment.Classic
```

```
{
```

```
    public class RabbitMQTransformer : ITransformer<ClassicDeployment, RabbitMQ.Model.Deployment>
```

```
    {
```

```
        //private readonly IDeploymentServerMapper _deploymentServerMapper;
```

```
        //private readonly ProjectRepository _projectRepository;
```

```
        //private readonly Regex _useDatabase = new Regex(@"USE\s+\[?(?<DatabaseName>\w+)\]?",
```

```
RegexOptions.IgnoreCase);
```

```
        //private readonly Regex _tableDDLStatement = new
```

```
Regex(@"(?:<Action>CREATE|ALTER|DROP)\s+TABLE\s+(dbo\.)?\[?(?<TableName>\w+)\]?",
```

```
RegexOptions.IgnoreCase);
```

```
        //private readonly Regex _projectRegex = new Regex(@"^[^/](?<ProjectName>[^/])");
```

```
        public RabbitMQTransformer()
```

```
{  
  
    //_deploymentServerMapper = new DeploymentServerMapperFactory().Create();  
  
    //_projectRepository = new ProjectRepository();  
  
}
```

```
public async Task<IEnumerable<RabbitMQ.Model.Deployment>> TransformAsync(  
  
    IEnumerable<ClassicDeployment> extracted,  
  
    CancellationToken cancellationToken,  
  
    IEtlProcessLogger logger)  
  
{  
  
    var results = await Task  
  
        .WhenAll(extracted.Select(cd =>  
  
            TransformAsync(cd, cancellationToken, logger)));  
  
    return results.SelectMany(a => a);  
  
}
```

```
private async Task<IEnumerable<RabbitMQ.Model.Deployment>> TransformAsync(  
  
    ClassicDeployment deployment,  
  
    CancellationToken cancellationToken,  
  
    IEtlProcessLogger logger)  
  
{  
  
    await Task.Yield();  
  
  
  
    var items = new List<RabbitMQ.Model.Deployment>();  
  
  
  
    items.Add(new RabbitMQ.Model.Deployment
```

```

{

Deployment_Key = deployment.Deployment_Key,

Copy_Of_Deployment_Key = deployment.Copy_Of_Deployment_Key,

First_Deployment_Key = deployment.First_Deployment_Key,

Deployment_Date = deployment.Deployment_Date,

//Commit_Date = deployment.Commit_Date,

Project_Name = deployment.Project_Name,

Deployment_Cart_Name = deployment.Cart_Name,

Database_Name = deployment.Database_Name,

Environment = deployment.Environment,

Owner = deployment.Owner,

Deployed_By = deployment.Deployed_By,

Rollback_Deployment = deployment.Rollback_Deployment,

Duplicate_Deployment = deployment.Duplicate_Deployment,

Deployment_Tool = "Classic Deployer",

Service_Tickets = deployment.Service_Tickets

});


//var sqlServers = deployment.Deployed_To_DB_Servers?.Split(',')?.Select(s => s.Trim()).ToList();

//var environments = deployment.Deployed_To_Webservers?.Split(',')?.Select(s => s.Replace("Deploy",
"").Trim()).ToList();


//string project = null;

//if (deployment.Deployed_To_Webservers?.EndsWith("Deploy") ?? false)

//{{

// project = "Classic";

```

```

//}

//else if (deployment.Filename.StartsWith("/Render", System.StringComparison.InvariantCultureIgnoreCase))

//{

// project = "Render";

//}

//else if (string.IsNullOrEmpty(deployment.Filename) == false)

//{

// project = _projectRegex.Match(deployment.Filename)?.Groups["ProjectName"]?.Value;

//}


//var webServers = new List<WebServer>();

//foreach (var environment in environments)

//{

// var outputs = await _projectRepository.GetBuildOutputs(project, environment);

// foreach (var output in await _projectRepository.GetBuildOutputs(project, environment))

// {

//   var bla = await _deploymentServerMapper.GetByShareName(output.SharePath);

//   webServers.AddRange(await _deploymentServerMapper.GetByShareName(output.SharePath));

// }

//}


//if (string.IsNullOrEmpty(deployment.Script) == false)

//{

// var databaseName = _useDatabase.Match(deployment.Script)?.Groups["DatabaseName"]?.Value;

// var matches = _tableDDLStatement.Matches(deployment.Script);

// foreach (var match in matches.OfType<Match>())

```

```

// {

//   var action = match.Groups["Action"]?.Value;

//   var tableName = match.Groups["TableName"]?.Value;

// }

//}


return items;

}

}

}

```

Repository.cs (Monitoring.ETL.Domain\Deployment\Classic\Repository.cs):

```

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Threading;

using System.Threading.Tasks;


namespace Monitoring.ETL.Domain.Deployment.Classic

{

    internal class Repository

    {

        public async Task<List<ClassicDeployment>> GetAllAfterDate(DateTime date, CancellationToken

cancellationToken)

        {

            var deployments = new List<ClassicDeployment>();

```

```

using (var connection = new SqlConnection(@"data source=AHPDS\AH_PDS;User
Id=PLEX_SIEM_READ_PDS;Password='2n5d;&r~VUEVN6';initial catalog=Development;"))

using (var command = connection.CreateCommand())

{

    var keyParam = command.CreateParameter();

    keyParam.ParameterName = "@Last_Deployment_Date";

    keyParam.DbType = System.Data.DbType.DateTime;

    keyParam.Direction = System.Data.ParameterDirection.Input;

    keyParam.Value = date;


    command.CommandType = System.Data.CommandType.Text;

    command.CommandText = @"

```

```

USE Development;

```

```

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

```

```

CREATE TABLE #Project_Folders

```

```

(
    Folder VARCHAR(100) NOT NULL PRIMARY KEY,
    Subfolder_Index AS LEN(Folder) + 1
);

```

```

CREATE TABLE #Environments

```

```

(
    Deployment_Target_Key INT NOT NULL,
    Environment VARCHAR(50) NOT NULL,

```

INDEX IX_Target CLUSTERED

```
(  
    Deployment_Target_Key  
)  
);
```

INSERT #Environments

VALUES

```
(1, 'Development'), --Development  
(3, 'Production'), --Production - Classic  
(3, 'Test'), --Production - Classic  
(4, 'Test'), --Test  
(5, 'Production'), --Production - Vision Plex  
(5, 'Test'), --Production - Vision Plex  
(6, 'Production'), --Report  
(10, 'Production'), --All Servers  
(10, 'Test'), --All Servers  
(10, 'Development'), --All Servers  
(14, 'Test'); --Test - Vision Plex
```

INSERT #Project_Folders

```
(  
    Folder  
)
```

VALUES

('/ClassicConnector/'),
('/ConsoleApplications/'),
('/ConsoleApplications/edi/'),
('/ConsoleApplications/fax/'),
('/ConsoleApplications/inventory/'),
('/ConsoleApplications/pacejet/'),
('/ConsoleApplications/perfmon/'),
('/ConsoleApplications/PlexScheduledJobs/'),
('/CustomerOwnedDataSources/'),
('/CustomerOwnedDataSources/Kors/'),
('/CustomerOwnedDataSources/RevolutionGroup/'),
('/CustomerOwnedDataSources/RevolutionGroup/Salesforce Wrapper DS/'),
('/directplex/'),
('/DotNetDeployer/'),
--('/edi/'),
('/edi/plex.edi/'),
('/ExchangeServer/'),
('/Execution/'),
('/FileWatcher/'),
('/Handlers/'),
('/Handlers/HoverMenu/'),
('/IIS/'),
('/JobManager/'),
('/Libraries/'),
('/MRPGeneration/'),
('/OpenSource/'),

('/OpenSource/Sphinx/'),
('/P2B2/'),
('/Pending_Notification/'),
('/PlexIdentityServer/'),
('/PlexWeb/'),
('/Plexus/'),
('/PlexusActivation/'),
('/PlexusCryptography/'),
('/PlexusDeployment/'),
('/PlexusDomain/'),
('/PlexusModules/'),
('/PlexusSqlClr/'),
('/PlexusWeb/'),
('/Plugins/'),
('/Plugins/LineManager/'),
('/PolAd/'),
('/RenderingModules/'),
('/Services/'),
('/Services/Pending_Notification/'),
('/Services/UniversalImport/'),
('/StaticContent/'),
('/ThirdParty/'),
('/RenderingEngine/'),
('/VisualStudioExtensions/'),
('/WebServices/');

WITH Deployments AS

(

SELECT TOP(100)

D.Deployment_Key,

D.Deployment_Date,

D.Deployed_To_DB_Servers,

D.Deployed_To_Webservers,

D.[Owner],

D.Deployed_By,

D.Cart_Name,

D.Rollback_Deployment,

D.Copy_Of_Deployment_Key,

E.Environment

FROM dbo.Deployment AS D

JOIN #Environments AS E

ON E.Deployment_Target_Key = D.Deployment_Target_Key

WHERE D.Deployment_Date > @Last_Deployment_Date

ORDER BY

D.Deployment_Date

),

Previous_Deployments AS

(

SELECT

Deployment_Key,

Copy_Of_Deployment_Key AS Previous_Deployment_Key,

Environment

FROM Deployments AS D

UNION ALL

SELECT

PD.Deployment_Key,

D.Copy_Of_Deployment_Key AS Previous_Deployment_Key,

PD.Environment

FROM Previous_Deployments AS PD

JOIN dbo.Deployment AS D

ON D.Deployment_Key = PD.Previous_Deployment_Key

),

Duplicate_Deployments AS

(

SELECT

D.Deployment_Key,

PD.Environment,

MIN(D2.Deployment_Key) AS First_Deployment_Key

FROM Previous_Deployments AS PD

JOIN dbo.Deployment AS D

ON D.Deployment_Key = PD.Deployment_Key

JOIN dbo.Deployment AS D2

ON D2.Deployment_Key = PD.Previous_Deployment_Key

JOIN #Environments AS E

ON E.Deployment_Target_Key = D2.Deployment_Target_Key

WHERE E.Environment = PD.Environment

GROUP BY

D.Deployment_Key,

PD.Environment

),

Deployment_Items AS

(

SELECT

DI.Deployment_Key,

DI.SQL_Script,

DI.Path_Filename,

DI.SQL_Object_Key,

DI.Snapshot_Date,

DI.Directed_Deployment_Plexus_Server_Group_Key,

LTRIM(

REPLACE(

REPLACE(

REPLACE(

REPLACE(

REPLACE(DI.Code,

'I, '),

'I, '),

',' '),

CHAR(10), ' '),

CHAR(13), ' ')

) AS Code

FROM Deployments AS D

JOIN dbo.Deployment_Item AS DI

ON DI.Deployment_Key = D.Deployment_Key

WHERE DI.Map_Checkout_Key IS NULL

AND DI.VP_Map = 0

)

SELECT DISTINCT

X.Deployment_Key,

--X.Commit_Date,

X.Deployment_Date,

X.Deployed_To_DB_Servers,

X.Directed_Deployment_Plexus_Server_Group_Key,

X.Deployed_To_Webservers,

X.Environment,

X.[Owner],

X.Deployed_By,

X.Copy_Of_Deployment_Key,

X.First_Deployment_Key,

X.Cart_Name,

X.Rollback_Deployment,

X.Duplicate_Deployment,

X.Multi_Line_Package_Deployment,

X.Project_Name,

X.Database_Name,

(

SELECT

*

FROM

(

SELECT

CONVERT(VARCHAR(10), DS.Support_Key) AS Service_Ticket_Key,

'USR' AS Service_Ticket_Type

FROM dbo.Deployment_Support AS DS

WHERE DS.Deployment_Key = X.Deployment_Key

UNION ALL

SELECT

DEI.External_Issue_Key AS Service_Ticket_Key,

'Jira' AS Service_Ticket_Type

FROM dbo.Deployment_External_Issue AS DEI

WHERE DEI.Deployment_Key = X.Deployment_Key

) AS X

FOR JSON AUTO

) AS Service_Tickets

FROM

(

SELECT

D.Deployment_Key,

DI.Snapshot_Date AS Commit_Date,

D.Deployment_Date,

D.Deployed_To_DB_Servers,

DI.Directed_Deployment_Plexus_Server_Group_Key,

```

D.Deployed_To_Webservers,

D.Environment,

D.[Owner],

D.Deployed_By,

D.Copy_Of_Deployment_Key,

ISNULL(PD.Base_Deployment_Cart_Key, D.Deployment_Key) AS First_Deployment_Key,

D.Cart_Name,

D.Rollback_Deployment,

CAST(CASE WHEN DD.First_Deployment_Key IS NOT NULL THEN 1 ELSE 0 END AS BIT) AS

Duplicate_Deployment,

CAST(0 AS BIT) AS Multi_Line_Package_Deployment,

DI.Path_Filename,

DI.Code,

CASE

    WHEN SO.SQL_Object_Key IS NOT NULL OR DI.SQL_Script = 1 THEN NULL

    WHEN CHARINDEX('/', DI.Path_Filename, PF.Subfolder_Index) = 0 THEN 'Classic'

    WHEN PF.Folder IS NOT NULL

        THEN SUBSTRING(DI.Path_Filename, PF.Subfolder_Index, CHARINDEX('/', DI.Path_Filename,

PF.Subfolder_Index) - PF.Subfolder_Index)

    ELSE 'Classic'

END AS Project_Name,

CASE

    WHEN SO.DB IS NOT NULL THEN SO.DB

    WHEN DI.SQL_Script = 1 AND DI.Code LIKE 'USE %' THEN

        SUBSTRING(DI.Code, 5, CHARINDEX(' ', DI.Code, 5) - 5)

END AS Database_Name

```


FROM Deployments AS D

JOIN Deployment_Items AS DI

ON DI.Deployment_Key = D.Deployment_Key

LEFT OUTER JOIN dbo.Sql_Object AS SO

ON SO.SQL_Object_Key = DI.SQL_Object_Key

LEFT OUTER JOIN Duplicate_Deployments AS DD

ON DD.Deployment_Key = D.Deployment_Key

AND DD.Environment = D.Environment

OUTER APPLY

(

SELECT

MIN(PD2.Previous_Deployment_Key) AS Base_Deployment_Cart_Key

FROM Previous_Deployments AS PD2

WHERE PD2.Deployment_Key = D.Deployment_Key

) AS PD

OUTER APPLY

(

SELECT TOP(1)

PF2.Folder,

PF2.Subfolder_Index

FROM #Project_Folders AS PF2

WHERE DI.Path_Filename LIKE PF2.Folder + '%'

ORDER BY

PF2.Subfolder_Index DESC

) AS PF

) AS X

WHERE X.Database_Name IS NOT NULL

OR X.Project_Name IS NOT NULL

DROP TABLE #Project_Folders;

DROP TABLE #Environments;"

```
command.Parameters.Add(keyParam);
```

```
await connection.OpenAsync();
```

```
var reader = await command.ExecuteReaderAsync();
```

```
if (reader.HasRows)
```

```
{
```

```
    var columns = new Dictionary<string, int>();
```

```
    for (int c = 0; c < reader.FieldCount; c++)
```

```
    {
```

```
        columns[reader.GetName(c)] = c;
```

```
    }
```

```
    while (await reader.ReadAsync(cancellationToken))
```

```
    {
```

```
        deployments.Add(new ClassicDeployment
```

```
        {
```

```
            Deployment_Key = await Get<int>(reader, columns["Deployment_Key"]),
```

```
            //Commit_Date = await Get<DateTime>(reader, columns["Commit_Date"]),
```

```
            Deployment_Date = await Get<DateTime>(reader, columns["Deployment_Date"]),
```

```

        Deployed_To_DB_Servers = await Get<string>(reader, columns["Deployed_To_DB_Servers"]),

        Directed_Deployment_Plexus_Server_Group_Key = await Get<int>(reader,

columns["Directed_Deployment_Plexus_Server_Group_Key"]),

        Deployed_To_Webservers = await Get<string>(reader, columns["Deployed_To_Webservers"]),

        Owner = await Get<int>(reader, columns["Owner"]),

        Deployed_By = await Get<int>(reader, columns["Deployed_By"]),

        Copy_Of_Deployment_Key = await Get<int>(reader, columns["Copy_Of_Deployment_Key"]),

        First_Deployment_Key = await Get<int>(reader, columns["First_Deployment_Key"]),

        //Classic_Application_Key = await Get<int>(reader, columns["Classic_Application_Key"]),

        //Filename = await Get<string>(reader, columns["Filename"]),

        //SQL_Object_Key = await Get<int>(reader, columns["SQL_Object_Key"]),

        //Script = await Get<string>(reader, columns["Script"]),

        Cart_Name = await Get<string>(reader, columns["Cart_Name"]),

        Rollback_Deployment = await Get<bool>(reader, columns["Rollback_Deployment"]),

        Duplicate_Deployment = await Get<bool>(reader, columns["Duplicate_Deployment"]),

        Multi_Line_Package_Deployment = await Get<bool>(reader,

columns["Multi_Line_Package_Deployment"]),

        Project_Name = await Get<string>(reader, columns["Project_Name"]),

        Database_Name = await Get<string>(reader, columns["Database_Name"]),

        Environment = await Get<string>(reader, columns["Environment"]),

        Service_Tickets = await Get<string>(reader, columns["Service_Tickets"])

    });

}

}

connection.Close();

```

```

    }

    return deployments;
}

private async Task<T> Get<T>(SqlDataReader reader, int ordinal)
{
    var value = reader.GetValue(ordinal);

    try
    {
        return (await reader.IsDBNullAsync(ordinal)) ? default : (T)reader.GetValue(ordinal);
    }

    catch (Exception)
    {
        return default;
    }
}
}
}

```

WebServerRepository.cs (Monitoring.ETL.Domain\Deployment\Classic\WebServerRepository.cs):

```

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.IO;

using System.Linq;

```

```
using System.Text.RegularExpressions;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.Deployment.Classic
```

```
{
```

```
    internal interface IDeploymentServerMapper
```

```
    {
```

```
        Task<IEnumerable<WebServer>> GetByShareName(string share);
```

```
    }
```

```
    internal class DeploymentServerMapperFactory
```

```
    {
```

```
        public IDeploymentServerMapper Create()
```

```
        {
```

```
            return new PeerSyncDeploymentServerMapper("Plex.snc");
```

```
        }
```

```
    }
```

```
    internal class PeerSyncDeploymentServerMapper : IDeploymentServerMapper
```

```
    {
```

```
        private readonly string _fileName;
```

```
        private readonly Regex _regex = new Regex(@"^\\w+\\(?:<DeployShare>.+\\)\\|\\\\\\(?:<WebServer>.+\\)\\");
```

```
        private ILookup<string, WebServer> _serversByGroup;
```

```
        public PeerSyncDeploymentServerMapper(string fileName)
```

```
        {
```

```

    _fileName = fileName;
}

public async Task<IEnumerable<WebServer>> GetByShareName(string sharePath)
{
    if (_serversByGroup == null)
    {
        await LoadServersFromFile();
    }

    return _serversByGroup
        .Where(g => sharePath.StartsWith(g.Key, StringComparison.InvariantCultureIgnoreCase))
        .SelectMany(g => g);
}

private async Task LoadServersFromFile()
{
    var results = new List<Tuple<string, string>>();

    using (var reader = new StreamReader(_fileName))
    {
        while (reader.EndOfStream == false)
        {
            var line = await reader.ReadLineAsync();

            var match = _regex.Match(line);

            var share = match?.Groups["DeployShare"]?.Value;

```

```
var server = match?.Groups["WebServer"]?.Value;
```

```
if (string.IsNullOrEmpty(share) == false && string.IsNullOrEmpty(server) == false)
```

```
{
```

```
    results.Add(new Tuple<string, string>(share.Replace("E:", ""), server));
```

```
}
```

```
}
```

```
}
```

```
_serversByGroup = results.ToLookup(t => t.Item1, t => new WebServer { Name = t.Item2 });
```

```
}
```

```
}
```

```
internal class ProjectEnvironmentBuildOutput
```

```
{
```

```
    public string Environment { get; set; }
```

```
    public string ProjectName { get; set; }
```

```
    public string SharePath { get; set; }
```

```
}
```

```
internal class ProjectRepository
```

```
{
```

```
    private const string ProjectSql = @"
```

```
USE Development;
```

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

SELECT

P.Project_Name,

PTT.Project_Target_Type AS Environment,

Target_Path AS Share_Path

FROM Development.dbo.Project_Target AS PT

JOIN dbo.Project AS P

ON P.Project_Key = PT.Project_Key

JOIN dbo.Project_Target_Type AS PTT

ON PTT.Project_Target_Type_Key = PT.Project_Target_Type_Key

WHERE Project_Target_Type IN ('Production', 'Test');";

private const string UncPathSql = @"

USE Plexus_Control;

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

SELECT

'Classic' AS Project_Name,

CASE P.[Name] WHEN 'Production Deploy' THEN 'Production' ELSE 'Test' END AS Environment,

P.Production_Path AS Target_Path

FROM Plexus_Control.dbo.UNC_Path AS P

WHERE P.[Name] IN ('Production Deploy', 'Test Deploy');";

private IEnumerable<ProjectEnvironmentBuildOutput> _buildOutputs;

public async Task<IEnumerable<ProjectEnvironmentBuildOutput>> GetBuildOutputs(string projectName, string environment)


```

{

    if (_buildOutputs == null)

    {

        await LoadProjectBuildOutputs();

    }

    return _buildOutputs.Where(o => o.ProjectName == projectName && o.Environment == environment);

}

private async Task LoadProjectBuildOutputs()

{

    _buildOutputs = (await LoadProjectBuildOutputs(ProjectSql)).Concat(await

LoadProjectBuildOutputs(UncPathSql));

    foreach (var buildOutput in _buildOutputs)

    {

        buildOutput.SharePath = buildOutput.SharePath.Replace("\\\\ah-deploy-vm", string.Empty);

    }

}

private async Task<IEnumerable<ProjectEnvironmentBuildOutput>> LoadProjectBuildOutputs(string sql)

{

    var buildOutputs = new List<ProjectEnvironmentBuildOutput>();

    using (var connection = new SqlConnection(@"data source=AH_N1_DEV\AH_N1_DEV;initial

catalog=Development;integrated security=True"))

```

```

using (var command = connection.CreateCommand())

{

    command.CommandType = System.Data.CommandType.Text;

    command.CommandText = sql;


    await connection.OpenAsync();

    var reader = await command.ExecuteReaderAsync();


    if (reader.HasRows)

    {

        while (await reader.ReadAsync())

        {

            buildOutputs.Add(new ProjectEnvironmentBuildOutput

            {

                ProjectName = reader.GetString(0),

                Environment = reader.GetString(1),

                SharePath = reader.GetString(2)

            });

        }

    }


    connection.Close();

}


return buildOutputs;

}

```

```
}
```

```
internal class WebServer
```

```
{
```

```
    public string Name { get; set; }
```

```
}
```

```
}
```

DelayFactory.cs (Monitoring.ETL.Domain\Deployment\DelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Deployment
```

```
{
```

```
    public class DelayFactory : ThresholdExtractionDelayFactory<RabbitMQ.Model.Deployment>
```

```
    {
```

```
        protected override int MaxThresholdForDelay => 100;
```

```
        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);
```

```
    }
```

```
}
```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\Deployment\RabbitMQExtractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Deployment
```

```
{  
  
    public sealed class RabbitMQExtractor : IExtractor<RabbitMQ.Model.Deployment>
```

```
{  
  
    private RabbitMQ.Deployment.Consumer _consumer;
```

```
  
    public async Task<ResultSet<RabbitMQ.Model.Deployment>> ExtractAllSinceLastExtractAsync(CancellationTok  
cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    _consumer = _consumer ?? new RabbitMQ.Deployment.Consumer(logger);  
  
    return await _consumer.ExtractAsync();  
  
}
```

```
  
    public Task<IEnumerable<RabbitMQ.Model.Deployment>> ExtractBetweenAsync(DateTime startDate, DateTime  
endDate, CancellationTok  
cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    throw new NotImplementedException();  
  
}
```

```
}
```

```
}
```

RabbitMQLoader.cs (Monitoring.ETL.Domain\Deployment\RabbitMQLoader.cs):

```
namespace Monitoring.ETL.Domain.Deployment
```

```

{

    public class RabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.Deployment>

    {

        public RabbitMQLoader()

            : base(new RabbitMQ.Deployment.Producer())

        {

        }

    }

}

```

RabbitMQWarehouseTransformer.cs (Monitoring.ETL.Domain\Deployment\RabbitMQWarehouseTransformer.cs):

```

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Domain.Warehouse.Model;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.Deployment

{

    public class RabbitMQWarehouseTransformer : ITransformer<RabbitMQ.Model.Deployment, Deployment_w>

    {

        public async Task<IEnumerable<Deployment_w>> TransformAsync(IEnumerable<RabbitMQ.Model.Deployment>

extracted, CancellationToken cancellationToken, IEtlProcessLogger logger)

```

```
{

    await Task.Yield();

    // TODO: Move Warehouse_Load_Add here to generate key?

    int i = 1;

    return extracted.Select(d =>

        new Deployment_w

        {

            Deployment_Key = d.Deployment_Key,

            Copy_Of_Deployment_Key = d.Copy_Of_Deployment_Key,

            First_Deployment_Key = d.First_Deployment_Key,

            Deployment_Date = d.Deployment_Date,

            Deployment_Cart_Name = d.Deployment_Cart_Name,

            Repository_Name = d.Repository_Name,

            Branch_Name = d.Branch_Name,

            Project_Name = d.Project_Name,

            Database_Name = d.Database_Name,

            Build_Name = d.Build_Name,

            Environment = d.Environment,

            Owner = d.Owner,

            Deployed_By = d.Deployed_By,

            Rollback_Deployment = d.Rollback_Deployment,

            Duplicate_Deployment = d.Duplicate_Deployment,

            Multi_Line_Package_Deployment = d.Multi_Line_Package_Deployment,

            Deployment_Tool = d.Deployment_Tool,

            Deployed_By_User_Id = d.Deployed_By_User_Id,

            Owner_User_Id = d.Owner_User_Id,
```

```

        Service_Tickets = d.Service_Tickets,

        Load_Event_Number = i++

    }).ToList();

}

}

}

```

DelayFactory.cs (Monitoring.ETL.Domain\Deployment\Smash\DelayFactory.cs):

```

using System;

namespace Monitoring.ETL.Domain.Deployment.Smash
{
    public class DelayFactory : ThresholdExtractionDelayFactory<SmashDeployment>
    {
        protected override int MaxThresholdForDelay => 100;

        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);
    }
}

```

Extractor.cs (Monitoring.ETL.Domain\Deployment\Smash\Extractor.cs):

```

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

```

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.ElasticSearch;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Deployment.Smash
```

```
{
```

```
    public class Extractor : IExtractor<SmashDeployment>
```

```
    {
```

```
        private readonly LoadStateRepository<SmashDeployment> _loadStateRepository;
```

```
        private readonly Repository _repository;
```

```
        private DateTime _lastDate;
```

```
        public Extractor()
```

```
        {
```

```
            _loadStateRepository = new LoadStateRepository<SmashDeployment>();
```

```
            _repository = new Repository();
```

```
        }
```

```
        public async Task<ResultSet<SmashDeployment>> ExtractAllSinceLastExtractAsync(CancellationTokencancellationTokentoken, IEtlProcessLogger logger)
```

```
        {
```

```
            var resultSet = new ResultSet<SmashDeployment>();
```

```
            try
```



```

{
    if (_lastDate == default)
    {
        _lastDate = _loadStateRepository.GetDate();
    }

    var deployments = await _repository.GetAllAfterDate(_lastDate, cancellationTokens);

    foreach (var deployment in deployments)
    {
        resultSet.Results.Add(deployment);
    }

    var max = deployments
        .Select(d => d.Deployment_Date)
        .OrderByDescending(date => date)
        .FirstOrDefault();

    if (max > _lastDate)
    {
        _lastDate = max;
        await _loadStateRepository.SetDate(_lastDate);
    }
}

catch (Exception ex)
{
    resultSet.Exceptions.Add(ex);
}

```

```

    }

    return resultSet;
}

public Task<IEnumerable<SmashDeployment>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
CancellationToken cancellationToken, IEtlProcessLogger logger)
{
    throw new NotImplementedException();
}
}
}
}

```

RabbitMQTransformer.cs (Monitoring.ETL.Domain\Deployment\Smash\RabbitMQTransformer.cs):

```

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.Deployment.Splash
{
    public class RabbitMQTransformer : ITransformer<SmashDeployment, RabbitMQ.Model.Deployment>
    {
        public async Task<IEnumerable<RabbitMQ.Model.Deployment>> TransformAsync(

```

```

IEnumerable<SmashDeployment> extracted,

CancellationToken cancellationToken,

IEtlProcessLogger logger)
{

    var results = await Task

        .WhenAll(extracted.Select(cd =>

            TransformAsync(cd, cancellationToken, logger)));

    return results.SelectMany(a => a);
}

private async Task<IEnumerable<RabbitMQ.Model.Deployment>> TransformAsync(

    SmashDeployment deployment,

    CancellationToken cancellationToken,

    IEtlProcessLogger logger)
{

    await Task.Yield();

    var items = new List<RabbitMQ.Model.Deployment>();

    items.Add(new RabbitMQ.Model.Deployment

    {

        Deployment_Key = deployment.Deployment_Plan_Execution_Key,

        Copy_Of_Deployment_Key = deployment.Previous_Deployment_Plan_Execution_Key,

        First_Deployment_Key = deployment.Job_Configuration_Key,

        Deployment_Date = deployment.Deployment_Date.ToLocalTime(),

        //Commit_Date = deployment.Commit_Date,

```

```

    Project_Name = deployment.Repository_Name,

    Deployment_Cart_Name = deployment.Cart_Name,

    //Database_Name = deployment.Database_Name,

    Environment = deployment.Environment.Split(' ').FirstOrDefault(),

    Owner_User_Id = deployment.Owner,

    Deployed_By_User_Id = deployment.Deployed_By,

    //Rollback_Deployment = deployment.Rollback_Deployment,

    Duplicate_Deployment = deployment.Duplicate_Deployment,

    Deployment_Tool = "SMASH",

    Service_Tickets = deployment.Service_Tickets

});

return items;

}

}

}

```

Repository.cs (Monitoring.ETL.Domain\Deployment\Smash\Repository.cs):

```

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

```

```

namespace Monitoring.ETL.Domain.Deployment.Splash

```

```

{

    internal class Repository

    {

        public async Task<List<SmashDeployment>> GetAllAfterDate(DateTime lastDate, CancellationToken

cancellationToken)

        {

            var deployments = new List<SmashDeployment>();

            using (var connection = new SqlConnection(@"data source=r162wzc7s5.database.windows.net,1433;User

Id=meta_readonly;Password='zDRLYb6FhNrjvwFWfizu5X3hXsVz4pF';initial catalog=Smash;"))

            using (var command = connection.CreateCommand())

            {

                var keyParam = command.CreateParameter();

                keyParam.ParameterName = "@Last_Date";

                keyParam.DbType = System.Data.DbType.DateTime;

                keyParam.Direction = System.Data.ParameterDirection.Input;

                keyParam.Value = lastDate;

                command.CommandType = System.Data.CommandType.Text;

                command.CommandText = @"

USE Smash;

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

SELECT TOP(100)

PE.Deployment_Plan_Execution_Key,

TE.Begin_Date AS Deployment_Date,

```

```

E.Name AS [Environment],

DBU.User_Name AS Deployed_By,

OU.User_Name AS Owner,

RC.Name AS Repository_Name,

JC.Name AS Cart_Name,

JC.Job_Configuration_Key,

CAST(CASE WHEN PPE.Deployment_Plan_Execution_Key IS NULL THEN 0 ELSE 1 END AS BIT) AS

Duplicate_Deployment,

PPE.Deployment_Plan_Execution_Key AS Previous_Deployment_Plan_Execution_Key,

CAST(0 AS BIT) AS Multi_Line_Package_Deployment,

(

SELECT

    I.External_Identity AS Service_Ticket_Key,

    CASE I.Issue_Tracker_Type_Key WHEN 1 THEN 'USR' WHEN 2 THEN 'Jira' END AS Service_Ticket_Type

FROM dbo.Job_Configuration_Issue AS JCI

JOIN dbo.Issue AS I

    ON I.Issue_Key = JCI.Issue_Key

WHERE JCI.Job_Configuration_Key = JC.Job_Configuration_Key

FOR JSON AUTO

) AS Service_Tickets

FROM dbo.Deployment_Task_Execution AS TE

JOIN dbo.[User] AS DBU

    ON DBU.User_Key = TE.Executed_By_User_Key

JOIN dbo.Deployment_Plan_Execution AS PE

    ON PE.Deployment_Plan_Execution_Key = TE.Deployment_Plan_Execution_Key

JOIN dbo.Deployment_Plan AS P

```

```

ON P.Deployment_Plan_Key = PE.Deployment_Plan_Key

JOIN dbo.Deployment_Environment AS E

ON E.Deployment_Environment_Key = P.Deployment_Environment_Key

JOIN dbo.Operation_Configuration AS OC

ON OC.Operation_Configuration_Key = P.Operation_Configuration_Key

JOIN dbo.Repository_Configuration AS RC

ON RC.Repository_Configuration_Key = OC.Repository_Configuration_Key

JOIN dbo.Job_Configuration AS JC

ON JC.Job_Configuration_Key = OC.Job_Configuration_Key

JOIN dbo.[User] AS OU

ON OU.User_Key = JC.Assigned_To_User_Key

OUTER APPLY

(

SELECT TOP (1)

PE2.Deployment_Plan_Execution_Key

FROM dbo.Deployment_Plan_Execution AS PE2

WHERE PE2.Deployment_Plan_Key = PE.Deployment_Plan_Key

AND PE2.Deployment_Plan_Execution_Key < PE.Deployment_Plan_Execution_Key

ORDER BY

PE2.Deployment_Plan_Execution_Key DESC

) AS PPE

WHERE TE.Begin_Date > @Last_Date

AND NOT EXISTS

(

SELECT

*

```

```

FROM dbo.Deployment_Task_Execution AS TE2

WHERE TE2.Deployment_Plan_Execution_Key = TE.Deployment_Plan_Execution_Key

AND TE2.Begin_Date < TE.Begin_Date

)

ORDER BY

TE.Begin_Date

OPTION (FORCE ORDER);";

```

```

command.Parameters.Add(keyParam);

```

```

await connection.OpenAsync();

```

```

var reader = await command.ExecuteReaderAsync();

```

```

if (reader.HasRows)

```

```

{

```

```

    var columns = new Dictionary<string, int>();

```

```

    for (int c = 0; c < reader.FieldCount; c++)

```

```

    {

```

```

        columns[reader.GetName(c)] = c;

```

```

    }

```

```

while (await reader.ReadAsync(cancellationToken))

```

```

{

```

```

    deployments.Add(new SmashDeployment

```

```

    {

```

```

        Deployment_Plan_Execution_Key = await Get<int>(reader,

```



```

columns["Deployment_Plan_Execution_Key"]),

    Deployment_Date = await Get<DateTime>(reader, columns["Deployment_Date"]),

    Environment = await Get<string>(reader, columns["Environment"]),

    Job_Configuration_Key = await Get<int>(reader, columns["Job_Configuration_Key"]),

    Deployed_By = await Get<string>(reader, columns["Deployed_By"]),

    Owner = await Get<string>(reader, columns["Owner"]),

    Repository_Name = await Get<string>(reader, columns["Repository_Name"]),

    Cart_Name = await Get<string>(reader, columns["Cart_Name"]),

    Duplicate_Deployment = await Get<bool>(reader, columns["Duplicate_Deployment"]),

    Previous_Deployment_Plan_Execution_Key = await Get<int>(reader,
columns["Previous_Deployment_Plan_Execution_Key"]),

    Multi_Line_Package_Deployment = await Get<bool>(reader,
columns["Multi_Line_Package_Deployment"]),

    Service_Tickets = await Get<string>(reader, columns["Service_Tickets"])

    });

}

}

connection.Close();

}

return deployments;

}

private async Task<T> Get<T>(SqlDataReader reader, int ordinal)

{

```

```

        var value = reader.GetValue(ordinal);

        try

        {

            return (await reader.IsDBNullAsync(ordinal)) ? default(T) : (T)reader.GetValue(ordinal);

        }

        catch (Exception)

        {

            return default;

        }

    }

}

```

SmashDeployment.cs (Monitoring.ETL.Domain\Deployment\Smash\SmashDeployment.cs):

```
using System;
```

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Deployment.Smash
```

```
{
```

```
    public class SmashDeployment : IExtractModel
```

```
    {
```

```
        public int Deployment_Plan_Execution_Key { get; set; }
```

```
        public DateTime Deployment_Date { get; set; }
```

```
        public string Environment { get; set; }
```

```
        public string Deployed_By { get; set; }
```

```
        public string Owner { get; set; }
```

```

public string Repository_Name { get; set; }

public string Cart_Name { get; set; }

public bool Duplicate_Deployment { get; set; }

public int Previous_Deployment_Plan_Execution_Key { get; set; }

public int First_Deployment_Plan_Execution_Key { get; set; }

public bool Multi_Line_Package_Deployment { get; set; }

public int Job_Configuration_Key { get; internal set; }

public string Service_Tickets { get; internal set; }

}

}

```

WarehouseLoader.cs (Monitoring.ETL.Domain\Deployment\WarehouseLoader.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```

namespace Monitoring.ETL.Domain.Deployment
{
    public class WarehouseLoader : Loader<Warehouse.Model.Deployment_w>
    {
        public WarehouseLoader()
            : base("Fact_Deployment_Add")
        {
        }
    }
}

```

RabbitMQDelayFactory.cs (Monitoring.ETL.Domain\Dimension\Module\RabbitExtractor\RabbitMQDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Dimension.Module.RabbitExtractor
```

```
{
```

```
    public class RabbitMQDelayFactory : ThresholdExtractionDelayFactory<RabbitMQ.Model.RabbitMQModule>
```

```
    {
```

```
        protected override int MaxThresholdForDelay => int.MaxValue;
```

```
        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);
```

```
    }
```

```
}
```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\Dimension\Module\RabbitExtractor\RabbitMQExtractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Dimension.Module.RabbitExtractor
```

```
{
```

```
    public sealed class RabbitMQExtractor : IExtractor<RabbitMQ.Model.RabbitMQModule>
```

```
    {
```

```
private RabbitMQ.Module.Consumer _consumer;
```

```
public async Task<ResultSet<RabbitMQ.Model.RabbitMQModule>>
```

```
ExtractAllSinceLastExtractAsync(Cancellation_token cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    _consumer = _consumer ?? new RabbitMQ.Module.Consumer(logger);  
  
    var result = await _consumer.ExtractAsync();  
  
    return result;  
  
}
```

```
public Task<IEnumerable<RabbitMQ.Model.RabbitMQModule>> ExtractBetweenAsync(DateTime startDate,
```

```
DateTime endDate, Cancellation_token cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    throw new NotImplementedException();  
  
}  
  
}  
  
}
```

RabbitMQWarehouseTransformer.cs

(Monitoring.ETL.Domain\Dimension\Module\RabbitExtractor\RabbitMQWarehouseTransformer.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.Warehouse.Model;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Dimension.Module.RabbitExtractor
```

```
{
```

```
    public class RabbitMQWarehouseTransformer : ITransformer<RabbitMQ.Model.RabbitMQModule, Module_w>
```

```
    {
```

```
        public async Task<IEnumerable<Module_w>> TransformAsync(IEnumerable<RabbitMQ.Model.RabbitMQModule>
```

```
extracted, CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
    {
```

```
        await Task.Yield();
```

```
        var result = extracted.Select(pc => new Module_w
```

```
        {
```

```
            JSON_Message = pc.JsonMessage
```

```
        });
```

```
        return result;
```

```
    }
```

```
}
```

```
}
```

WarehouseLoader.cs (Monitoring.ETL.Domain\Dimension\Module\RabbitExtractor\WarehouseLoader.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.Dimension.Module.RabbitExtractor

{

    public class WarehouseLoader : Loader<Warehouse.Model.Module_w>

    {

        public WarehouseLoader() : base("Dim_Module_Add")

        {

        }

    }

}
```

DelayFactory.cs (Monitoring.ETL.Domain\Dimension\Module\RabbitLoader\DelayFactory.cs):

```
using System;
```

```
using Monitoring.ETL.Domain.Dimension.Module;
```

```
namespace Monitoring.ETL.Domain.Dimension.Module.RabbitLoader

{

    public class DelayFactory : TimeOfDayExtractionDelayFactory<ModuleJson>

    {

        protected override TimeSpan TimeOfDay => new TimeSpan(3, 30, 00);

    }

}
```

Extractor.cs (Monitoring.ETL.Domain\Dimension\Module\RabbitLoader\Extractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Configuration;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.ElasticSearch;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Dimension.Module.RabbitLoader
```

```
{
```

```
    /// <summary>
```

```
    /// Extract Dim_Module information from N Scale and place it into ModuleJson object
```

```
    /// </summary>
```

```
    public class Extractor : IExtractor<ModuleJson>
```

```
    {
```

```
        private readonly SqlJson.Repository<ModuleJson> _repository;
```

```
        public Extractor()
```

```
        {
```

```
            const string sql = @"SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;" +
```

```
                "SELECT " +
```

```
                " M.Module_Key," +
```

```
                " M.Name," +
```



```

" MG.Module_Group," +

" M.Internal_Only," +

" M.Standard_Module," +

" M.Required AS Required_Module," +

"    MS.Obsolete_Status AS Obsolete_Module," +

"    MS.Deploy_Allow," +

" MS.Module_Status," +

" '1900-01-01' AS Effective_Date," +

" '2100-12-31 23:59:59.997' AS Expiration_Date," +

" 1 AS Effective" +

" FROM dbo.Module AS M" +

" JOIN dbo.Module_Group AS MG" +

" ON MG.Module_Group_Key = M.Module_Group_Key" +

" JOIN dbo.Module_Status AS MS" +

" ON MS.Module_Status_Key = M.Module_Status_Key order by m.module_key for json path";

```

```

_repository = new SqlJson.Repository<ModuleJson>(
    new List<string> { "n1" },
    "Plexus_Control",
    sql
);
}

```

```

public Task<ResultSet<ModuleJson>> ExtractAllSinceLastExtractAsync(Cancellation token cancellationToken,
    IEtlProcessLogger logger)
{

```

```

        return _repository.ExtractAllSinceLastExtractAsync(cancellationToken, logger);
    }

    public Task<IEnumerable<ModuleJson>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
CancellationTok
en cancellationToken, IEtlProcessLogger logger)
    {
        throw new NotImplementedException();
    }
}
}
}

```

ModuleJson.cs (Monitoring.ETL.Domain.Dimension.Module.RabbitLoader\ModuleJson.cs):

```

using ETL.Process;

using Monitoring.ETL.Domain.SqlJson;

using System;

namespace Monitoring.ETL.Domain.Dimension.Module.RabbitLoader
{
    public class ModuleJson : IExtractModel, IJsonExtractModel
    {
        public string JsonMessage { get; set; }
    }
}

```

RabbitMQLoader.cs (Monitoring.ETL.Domain.Dimension.Module.RabbitLoader\RabbitMQLoader.cs):

```
namespace Monitoring.ETL.Domain.Dimension.Module.RabbitLoader
```

```
{  
  
    public class RabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.RabbitMQModule>  
  
    {  
  
        public RabbitMQLoader() : base(new RabbitMQ.Module.Producer())  
  
        {  
  
        }  
  
    }  
  
}
```

Transformer.cs (Monitoring.ETL.Domain.Dimension.Module.RabbitLoader\Transformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Dimension.Module.RabbitLoader
```

```
{  
  
    public class Transformer : ITransformer<ModuleJson, RabbitMQ.Model.RabbitMQModule>  
  
    {  
  
        public async Task<IEnumerable<RabbitMQ.Model.RabbitMQModule>>
```

```

TransformAsync(IEnumerable<ModuleJson> extracted,

    CancellationToken cancellationToken, IEtlProcessLogger logger)

{

    await Task.Yield();

    return extracted.Select(pc =>

        new RabbitMQ.Model.RabbitMQModule

        {

            JsonMessage = pc.JsonMessage

        });

    }

}

```

DelayFactory.cs (Monitoring.ETL.Domain.Dimension.Procedure.DelayFactory.cs):

```
using System;
```

```

namespace Monitoring.ETL.Domain.Dimension.Procedure

{

    public class DelayFactory : TimeOfDayExtractionDelayFactory<ProcedureInfo>

    {

        protected override TimeSpan TimeOfDay => new TimeSpan(3, 30, 00);

    }

}

```

Extractor.cs (Monitoring.ETL.Domain.Dimension.Procedure.Extractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Dimension.Procedure
```

```
{
```

```
    public class Extractor : IExtractor<ProcedureInfo>
```

```
    {
```

```
        private readonly SqlJson.Repository<ProcedureInfo> _repository;
```

```
        public Extractor()
```

```
        {
```

```
            _repository = new SqlJson.Repository<ProcedureInfo>(
```

```
                new List<string> { "n1" },
```

```
                "Plexus_Control",
```

```
                @"
```

```
USE Plexus_Control;
```

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

```
SELECT
```

```
    P.Procedure_Key,
```

```
    P.[Procedure_Name],
```

```
    D.[Database_Name]
```

```

FROM Plexus_Control.dbo.Procedure_Info AS P

JOIN Plexus_System.dbo.Plexus_Database AS D

    ON D.Plexus_Database_Key = P.Database_Key

FOR JSON PATH;");

    }

```

```

    public Task<ResultSet<ProcedureInfo>> ExtractAllSinceLastExtractAsync(Cancellation_token cancellationToken,
        IEtlProcessLogger logger)

    {

        return _repository.ExtractAllSinceLastExtractAsync(cancellationToken, logger);

    }

```

```

    public Task<IEnumerable<ProcedureInfo>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
        Cancellation_token cancellationToken, IEtlProcessLogger logger)

    {

        throw new NotImplementedException();

    }

}

```

ProcedureInfo.cs (Monitoring.ETL.Domain\Dimension\Procedure\ProcedureInfo.cs):

```

using ETL.Process;

namespace Monitoring.ETL.Domain.Dimension.Procedure

{

    public class ProcedureInfo : IExtractModel, SqlJson.IJsonExtractModel

```

```
{  
  
    public string JsonMessage { get; set; }  
  
}  
  
}
```

RabbitMQDelayFactory.cs (Monitoring.ETL.Domain\Dimension\Procedure\RabbitMQDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Dimension.Procedure
```

```
{  
  
    public class RabbitMQDelayFactory : ThresholdExtractionDelayFactory<RabbitMQ.Model.Procedure>  
  
    {  
  
        protected override int MaxThresholdForDelay => int.MaxValue;  
  
  
        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);  
  
    }  
  
}
```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\Dimension\Procedure\RabbitMQExtractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Dimension.Procedure
```

```
{
```

```
    public sealed class RabbitMQExtractor : IExtractor<RabbitMQ.Model.Procedure>
```

```
    {
```

```
        private RabbitMQ.Procedure.Consumer _consumer;
```

```
        public async Task<ResultSet<RabbitMQ.Model.Procedure>> ExtractAllSinceLastExtractAsync(CancellationTokencancellationTokentoken, IEtlProcessLogger logger)
```

```
        {
```

```
            _consumer = _consumer ?? new RabbitMQ.Procedure.Consumer(logger);
```

```
            return await _consumer.ExtractAsync();
```

```
        }
```

```
        public Task<IEnumerable<RabbitMQ.Model.Procedure>> ExtractBetweenAsync(DateTime startDate, DateTimeendDate, CancellationTokentoken, IEtlProcessLogger logger)
```

```
        {
```

```
            throw new NotImplementedException();
```

```
        }
```

```
    }
```

```
}
```

RabbitMQLoader.cs (Monitoring.ETL.Domain\Dimension\Procedure\RabbitMQLoader.cs):

```
namespace Monitoring.ETL.Domain.Dimension.Procedure
```

```
{
```

```
    public class RabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.Procedure>
```

```
    {
```



```

public RabbitMQLoader()

    : base(new RabbitMQ.Procedure.Producer())

{

}

}

}

```

RabbitMQWarehouseTransformer.cs

(Monitoring.ETL.Domain\Dimension\Procedure\RabbitMQWarehouseTransformer.cs):

```

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.Dimension.Procedure

{

    public class RabbitMQWarehouseTransformer : ITransformer<RabbitMQ.Model.Procedure,

Warehouse.Model.Procedure_w>

    {

        public async Task<IEnumerable<Warehouse.Model.Procedure_w>>

TransformAsync(IEnumerable<RabbitMQ.Model.Procedure> extracted, CancellationToken cancellationToken,

IEtlProcessLogger logger)

        {

```

```
await Task.Yield();
```

```
return extracted.Select(pc =>  
    new Warehouse.Model.Procedure_w  
    {  
        JSON_Message = pc.JsonMessage  
    });  
}  
  
}  
  
}
```

Transformer.cs (Monitoring.ETL.Domain\Dimension\Procedure\Transformer.cs):

```
using System.Collections.Generic;  
  
using System.Linq;  
  
using System.Threading;  
  
using System.Threading.Tasks;  
  
using ETL.Process;  
  
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Dimension.Procedure
```

```
{  
  
    public class Transformer : ITransformer<ProcedureInfo, RabbitMQ.Model.Procedure>  
    {  
  
        public async Task<IEnumerable<RabbitMQ.Model.Procedure>> TransformAsync(IEnumerable<ProcedureInfo>  
extracted, CancellationToken cancellationToken, IEtlProcessLogger logger)  
  
        {
```

```
await Task.Yield();
```

```
return extracted.Select(pc =>
```

```
    new RabbitMQ.Model.Procedure
```

```
{
```

```
    JsonMessage = pc.JsonMessage
```

```
});
```

```
}
```

```
}
```

```
}
```

WarehouseLoader.cs (Monitoring.ETL.Domain\Dimension\Procedure\WarehouseLoader.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.Dimension.Procedure
```

```
{
```

```
    public class WarehouseLoader : Loader<Warehouse.Model.Procedure_w>
```

```
{
```

```
    public WarehouseLoader()
```

```
    : base("Dim_Procedure_Add")
```

```
{
```

```
}
```

```
}
```

```
}
```

EDIUsageModel.cs (Monitoring.ETL.Domain\EDIUsage\Model\EDIUsageModel.cs):

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Monitoring.ETL.Domain.EDIUsage.Model

{

public class EDIUsageModel

{

/// <summary>

/// The Type of Data EDI_Usage_Data_Get SPROC Returns.

/// </summary>

public string Type { get; set; }

/// <summary>

/// The PCN Information

/// </summary>

public int PCN { get; set; }

/// <summary>

/// The Primary Key of EDI DB's Mailbox Table.

/// </summary>

public int MailboxKey { get; set; }

/// <summary>

/// The Primary Key of EDI DB's Document Table.

/// </summary>

public int DocumentKey { get; set; }

/// <summary>

/// The Primary Key of EDI DB's Template Table.

/// </summary>

public int TemplateKey { get; set; }

/// <summary>

/// The Primary Key of EDI DB's Trading_Partner Table.

/// </summary>

public int TradingPartnerKey { get; set; }

/// <summary>

/// The Date Information When Template Added to the Mailbox.

/// </summary>

public string TemplateAddDate { get; set; }

/// <summary>

/// The EDI Log Count Information

/// </summary>

public int Count { get; set; }

/// <summary>

/// The TransactionDate Information

/// </summary>

public DateTime TransactionDate { get; set; }

/// <summary>

/// The EDI Document's Name Information

/// </summary>

public string DocumentName { get; set; }

/// <summary>

/// The EDI Document's Type Information

/// </summary>

public string DocumentType { get; set; }

/// <summary>

/// The EDI Template's Name Information

/// </summary>

public string TemplateName { get; set; }

/// <summary>

/// The EDI Template's Associate Stored Procedure Information

/// </summary>

public string TemplateStoredProcedures { get; set; }

/// <summary>

/// The EDI Mailbox's Name Information

/// </summary>

public string MailboxName { get; set; }

/// <summary>

/// The EDI Mailbox's MailboxCode Information

/// </summary>

public string MailboxCode { get; set; }

/// <summary>

/// The EDI Mailbox's DestinationMailbox Information

/// </summary>

public string DestinationMailbox { get; set; }

/// <summary>

/// The EDI Mailbox's DestinationCode Information

/// </summary>

public string DestinationCode { get; set; }

/// <summary>

/// The EDI Mailbox's DestinationQualifier Information

/// </summary>

public string DestinationQualifier { get; set; }

/// <summary>

/// The EDI Mailbox's HondaPlantCode Information

/// </summary>

public string HondaPlantCode { get; set; }

/// <summary>

/// The EDI Mailbox's DestinationInternalId Information

/// </summary>

public string DestinationInternalId { get; set; }

/// <summary>

/// The EDI Mailbox's ElementSeperator Information

/// </summary>

public string ElementSeparator { get; set; }

/// <summary>

/// The EDI Mailbox's SubElementSeperator Information

/// </summary>

public string SubelementSeparator { get; set; }

/// <summary>

/// PCN Data for the Mailbox

/// </summary>

public int MailboxPCN { get; set; }

/// <summary>

/// The EDI Mailbox's SegmentTerminator Information

/// </summary>


```
public int SegmentTerminator { get; set; }
```

```
/// <summary>
```

```
/// The EDI Mailbox's MailboxStatus Information
```

```
/// </summary>
```

```
public string MailboxStatus { get; set; }
```

```
/// <summary>
```

```
/// The When EDI Mailbox Added Information
```

```
/// </summary>
```

```
public string MailboxAddDate { get; set; }
```

```
/// <summary>
```

```
/// The When EDI Mailbox Updated Information
```

```
/// </summary>
```

```
public string MailboxUpdateDate { get; set; }
```

```
/// <summary>
```

```
/// The When EDI Mailbox's VANConnectionName Information
```

```
/// </summary>
```

```
public string VANConnectionName { get; set; }
```

```
/// <summary>
```

```
/// The When EDI Mailbox's ValueAddedNetwork Information
```

```
/// </summary>
```

```
public string ValueAddedNetwork { get; set; }
```

```

    /// <summary>

    /// The When Trading Parter Information

    /// </summary>

    public string TradingPartnerName { get; set; }

}

}

DelayFactory.cs (Monitoring.ETL.Domain\EDIUsage\RabbitMQLoader\DelayFactory.cs):

using ETL.Process;

using Monitoring.ETL.Process;

using System;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

namespace Monitoring.ETL.Domain.EDIUsage.RabbitMQLoader

{

    public class DelayFactory : IExtractionDelayFactory<EDIUsageDataModel>

    {

        private readonly TimeSpan _runAtTimespan = new TimeSpan(3, 30, 00);

        private TimeSpan _retryDelayTimespan = new TimeSpan(0, 0, 5);

```

```

public async Task Create(ResultSet<EDIUsageDataModel> results, CancellationToken cancellationToken,
IEtlProcessLogger logger)
{
    var yesterday = DateTime.Today.AddDays(-1);

    // Look for exceptions other than empty results exception and if found, retry extraction in 10 minutes
    if (results.Exceptions.Any(e => !(e is EmptyResultsException)))
    {
        logger.Information($"Incomplete extraction detected... Retrying in {_retryDelayTimespan.Minutes} minutes.");
        await Task.Delay(_retryDelayTimespan, cancellationToken);

        return;
    }

    // Extraction didn't yield any results, see if the last extraction date is yesterday, if not, skip delay
    if (results.Exceptions.Count == 1 && results.Exceptions[0] is EmptyResultsException emptyResultsException &&
        emptyResultsException.TransactionDate < yesterday)
    {
        logger.Information($"Skipping delay until extracted 'Transaction date'
({emptyResultsException.TransactionDate:yyyy-MM-dd}) is yesterday ({yesterday:yyyy-MM-dd}).");

        return;
    }

    // Results found! If any result having transaction date less than yesterday is found, skip delay to pull more results
    if (results.Results.Any(r => r.TransactionDate < yesterday))
    {

```

```

        logger.Information($"Skipping delay until extracted 'Transaction date' ({results.Results.Min(r =>
r.TransactionDate):yyyy-MM-dd}) is yesterday ({yesterday:yyyy-MM-dd}).");

        return;
    }

    // All extractions completed through yesterday, your job is done... Sleep till tomorrow...

    var nextRunTime = DateTime.Today.AddDays(1).Add(_runAtTimespan);

    logger.Information($"Extraction completed through yesterday... Sleeping until {nextRunTime}.");

    await Task.Delay(nextRunTime.Subtract(DateTime.Now), cancellationToken);

}

}

}

```

EDIUsageDataModel.cs (Monitoring.ETL.Domain\EDIUsage\RabbitMQLoader\EDIUsageDataModel.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.SqlJson;
```

```
using System;
```

```
namespace Monitoring.ETL.Domain.EDIUsage.RabbitMQLoader
```

```
{
```

```
    /// <summary>
```

```
    /// A model which holds the data which you're loading from your source. You will need to convert
```

```
    /// your data into a Json object which populates the JsonMessage property. This JSON object
```

```
    /// will then be loaded into RabbitMQ

```

```

/// </summary>

public class EDIUsageDataModel : IExtractModel, IJsonExtractModel, ILoadModel
{
    /// <summary>

    /// The transaction date on which Json data was extracted

    /// </summary>

    public DateTime TransactionDate { get; set; }


    /// <summary>

    /// The data to load, in a JSON Object

    /// </summary>

    public string JsonMessage { get; set; }

}
}

```

EDIUsageRabbitMqLoader.cs (Monitoring.ETL.Domain\EDIUsage\RabbitMQLoader\EDIUsageRabbitMqLoader.cs):

```

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;


namespace Monitoring.ETL.Domain.EDIUsage.RabbitMQLoader
{
    /// <summary>

    /// Wrapper class

```

/// </summary>

```
public sealed class EDIUsageRabbitMqLoader : RabbitMQ.Loader<RabbitMQ.Model.RabbitMQEDIUsageModel>

{

    public EDIUsageRabbitMqLoader() : base(new RabbitMQ.EDIUsage.Producer())

    {

    }

}

}
```

EmptyResultsException.cs (Monitoring.ETL.Domain\EDIUsage\RabbitMQLoader\EmptyResultsException.cs):

using System;

namespace Monitoring.ETL.Domain.EDIUsage.RabbitMQLoader

```
{

    public class EmptyResultsException : Exception

    {

        public EmptyResultsException(DateTime transactionDate) : base()

        {

            TransactionDate = transactionDate;

        }

        public DateTime TransactionDate { get; }

    }

}
```

NScaleExtractor.cs (Monitoring.ETL.Domain\EDIUsage\RabbitMQLoader\NScaleExtractor.cs):

```
using ETL.Process;

using Monitoring.ETL.Domain.EDIUsage.Model;

using Monitoring.ETL.Domain.ElasticSearch;

using Monitoring.ETL.Process;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using static Monitoring.ETL.Domain.ServiceToolConnectionRepository;
```

```
namespace Monitoring.ETL.Domain.EDIUsage.RabbitMQLoader
```

```
{

    public class NScaleExtractor : IExtractor<EDIUsageDataModel>

    {

        private readonly ServiceToolConnectionRepository _serviceToolConnectionRepository;

        private readonly LoadStateRepository _loadStateRepository;

        private readonly DateTime _defaultLoadDate;

        private const string ServiceToolName = "EDI Usage ETL";

        private const string UsageCountInfo = "UsageCountDetail";

        private const string DocumentInfo = "DocumentDetail";

        private const string TemplateInfo = "TemplateDetail";

    }

}
```

```
private const string MailboxInfo = "MailboxDetail";
```

```
private const string TradingPartnerInfo = "TradingPartnerDetail";
```

```
public NScaleExtractor()
```

```
{
```

```
    _loadStateRepository = new LoadStateRepository();
```

```
    _serviceToolConnectionRepository = new ServiceToolConnectionRepository();
```

```
    _defaultLoadDate = DateTime.TryParse(ConfigurationManager.AppSettings["EDI_Usage_Start_Date"], out  
DateTime date) ? date : new DateTime(2001, 1, 1).AddDays(-1);
```

```
}
```

```
public async Task<ResultSet<EDIUsageDataModel>> ExtractAllSinceLastExtractAsync(CancellationTok  
cancellationToken, IEtlProcessLogger logger)
```

```
{
```

```
    var resultSet = new ResultSet<EDIUsageDataModel>();
```

```
    var usageCounts = new List<EDIUsageDataModel>();
```

```
    var documents = new List<EDIUsageDataModel>();
```

```
    var templates = new List<EDIUsageDataModel>();
```

```
    var mailboxes = new List<EDIUsageDataModel>();
```

```
    var tradingPartners = new List<EDIUsageDataModel>();
```

```
    var loadStateUpdates = new Dictionary<string, DateTime>();
```

```
    DateTime lastLoadDate = _defaultLoadDate;
```

```
try
```

```
{
```

```
    logger.Information("Retrieving Servers to process");
```



```
var servers = _serviceToolConnectionRepository.GetServersToProcess(ServiceToolName).OrderBy(kvp =>
kvp.Key).ToList();
```

```
logger.Information($"{servers.Count} server(s) found");
```

```
foreach (var server in servers)
```

```
{
```

```
var serviceName = server.Key;
```

```
var sqlServerModel = server.Value;
```

```
try
```

```
{
```

```
logger.Information($"Processing SQL Server Instance: {serviceName}
```

```
{sqlServerModel.MachineName}");
```

```
string loadName = $"EDIUsage: {serviceName}";
```

```
lastLoadDate = GetLastLoadDate(loadName);
```

```
#if DEBUG
```

```
logger.Information($"Last load for {loadName} is {lastLoadDate} on
```

```
{_loadStateRepository.GetConnection()}");
```

```
#endif
```

```
var dateToLoad = lastLoadDate.AddDays(1);
```

```
// We want to load the data through yesterday...
```

// If "Last Load Date" is yesterday, we are done with extraction. We will defer execution till tomorrow so that we get all of "Today's" data...

```
if (dateToLoad.Date == DateTime.Today)
{
    logger.Information($"Skipping extraction for '{serviceName}' as it's already processed through
'{lastLoadDate:yyyy-MM-dd}'.");
    continue;
}

logger.Information($"Populating result sets for '{loadName}': Start");
// Populate EDI Usage data
// This needs to happen across all 'n's
PopulateEDIUsageResultSets(sqlServerModel, dateToLoad, ref usageCounts, ref documents, ref
templates, ref mailboxes, ref tradingPartners, logger);

logger.Information($"Populating result sets for '{loadName}': End");

// Update the "Last Load Date"
loadStateUpdates.Add(loadName, dateToLoad);
}
catch (Exception e)
{
    if (e.IsNetworkConnectionException())
    {
        logger.Information($"'{serviceName}' is currently unavailable. Skipping extraction for this execution.");
        continue;
    }
}
```

```
}
```

```
logger.Information($"An error occurred while extracting data from {serviceName}");
```

```
e.Data.Add("X-Server", server);
```

```
resultSet.Exceptions.Add(e);
```

```
}
```

```
}
```

```
documents.ForEach(resultSet.Results.Add);
```

```
templates.ForEach(resultSet.Results.Add);
```

```
mailboxes.ForEach(resultSet.Results.Add);
```

```
tradingPartners.ForEach(resultSet.Results.Add);
```

```
// ** ALWAYS ADD THE USAGE COUNTS RESULTS AFTER ALL THE META DATA TO AVOID BATCHING
```

```
ISSUES ** //
```

```
usageCounts.ForEach(resultSet.Results.Add);
```

```
// After all servers have been processed, persist the keys to the load states.
```

```
foreach (var loadStateUpdate in loadStateUpdates)
```

```
{
```

```
    await _loadStateRepository.SetStateAsync(loadStateUpdate.Key, loadStateUpdate.Value);
```

```
}
```

```
if (!resultSet.Results.Any())
```

```
{
```

```
    resultSet.Exceptions.Add(new EmptyResultsException(loadStateUpdates.Values.Any() ?
```

```

        loadStateUpdates.Values.Min() : lastLoadDate));

    }

}

catch (Exception e)

{

    logger.Information("An error occurred while extracting EDI usage data.");

    resultSet.Exceptions.Add(e);

    return resultSet;

}

return resultSet;

}

```

```

private DateTime GetLastLoadDate(string loadName)

{

    var lastLoadDate = _loadStateRepository.GetDateFor(loadName);

    if (lastLoadDate < _defaultLoadDate)

        lastLoadDate = _defaultLoadDate;

    return lastLoadDate;

}

```

```

private void PopulateEDIUsageResultSets(SqlServerModel sqlServerModel, DateTime dateToLoad, ref
List<EDIUsageDataModel> usageCounts, ref List<EDIUsageDataModel> documents, ref List<EDIUsageDataModel>
templates, ref List<EDIUsageDataModel> mailboxes, ref List<EDIUsageDataModel> tradingPartners, IEtlProcessLogger

```

logger)

```
{  
  
    var serverName = $"{sqlServerModel.MachineName} {(sqlServerModel.Port > 0 ? $":{sqlServerModel.Port}" :  
    $"{sqlServerModel.InstanceName}"}";  
  
    using (var con = new SqlConnection($"data source={serverName};initial catalog=EDI;integrated  
security=True;MultipleActiveResultSets=True;"))  
  
    {  
  
        con.Open();  
  
  
  
        using (var cmd = new SqlCommand("EDI.dbo.EDI_Usage_Data_Get", con))  
  
        {  
  
            cmd.CommandType = CommandType.StoredProcedure;  
  
            cmd.Parameters.Add(  
  
                new SqlParameter  
  
                {  
  
                    ParameterName = "@Usage_Date",  
  
                    SqlDbType = SqlDbType.DateTime,  
  
                    Value = dateToLoad  
  
                });  
  
  
  
            using (var reader = cmd.ExecuteReader())  
  
            {  
  
                // Result Set 1 - Usage Counts  
  
                if (reader.HasRows)  
  
                {  
  
                    var count = 0;
```

```
var pcnUCIOrdinal = reader.GetOrdinal("PCN");

var mailboxKeyUCIOrdinal = reader.GetOrdinal("Mailbox_Key");

var documentKeyUCIOrdinal = reader.GetOrdinal("Document_Key");

var templateKeyUCIOrdinal = reader.GetOrdinal("Template_Key");

var tradingpartnerKeyUCIOrdinal = reader.GetOrdinal("Trading_Partner_Key");

var templateAddDateUCIOrdinal = reader.GetOrdinal("Template_Add_Date");

var countUCIOrdinal = reader.GetOrdinal("EDI_Count");
```

```
while (reader.Read())
```

```
{
```

```
    count++;
```

```
    usageCounts.Add(new EDIUsageDataModel
```

```
    {
```

```
        TransactionDate = dateToLoad,
```

```
        JsonMessage = JsonConvert.SerializeObject(new EDIUsageModel
```

```
        {
```

```
            TransactionDate = dateToLoad,
```

```
            Type = UsageCountInfo,
```

```
            PCN = reader.GetInt32(pcnUCIOrdinal),
```

```
            MailboxKey = reader.GetInt32(mailboxKeyUCIOrdinal),
```

```
            DocumentKey = reader.GetInt32(documentKeyUCIOrdinal),
```

```
            TemplateKey = reader.GetInt32(templateKeyUCIOrdinal),
```

```
            TradingPartnerKey = reader.GetInt32(tradingpartnerKeyUCIOrdinal),
```

```
            TemplateAddDate = reader.GetString(templateAddDateUCIOrdinal),
```

```
            Count = reader.GetInt32(countUCIOrdinal),
```

```
        })
```

```

    });

}

logger.Information($"Usage Counts: {count}");

}

// Result Set 2 - Documents

if (reader.NextResult())

{

    var count = 0;

    var documentKeyDIOrdinal = reader.GetOrdinal("Document_Key");

    var documentNameDIOrdinal = reader.GetOrdinal("Document_Name");

    var documentTypeDIOrdinal = reader.GetOrdinal("Document_Type");

    while (reader.Read())

    {

        count++;

        documents.Add(new EDIUsageDataModel

        {

            TransactionDate = dateToLoad,

            JsonMessage = JsonConvert.SerializeObject(new EDIUsageModel

            {

                TransactionDate = dateToLoad,

                Type = DocumentInfo,

                DocumentKey = reader.GetInt32(documentKeyDIOrdinal),

                DocumentName = reader.GetString(documentNameDIOrdinal),

                DocumentType = reader.GetString(documentTypeDIOrdinal)
            })
        })
    }
}

```

```

        })

    });

}

logger.Information($"Documents: {count}");

}

// Result Set 3 - Templates

if (reader.NextResult())

{

    var count = 0;

    var templateKeyTIOrdinal = reader.GetOrdinal("Template_Key");

    var templateNameTIOrdinal = reader.GetOrdinal("Template_Name");

    var templateStoredProceduresTIOrdinal = reader.GetOrdinal("Template_Stored_Procedures");

    while (reader.Read())

    {

        count++;

        templates.Add(new EDIUsageDataModel

        {

            TransactionDate = dateToLoad,

            JsonMessage = JsonConvert.SerializeObject(new EDIUsageModel

            {

                TransactionDate = dateToLoad,

                Type = TemplateInfo,

                TemplateKey = reader.GetInt32(templateKeyTIOrdinal),

                TemplateName = reader.GetString(templateNameTIOrdinal),

```



```

        TemplateStoredProcedures = reader.GetString(templateStoredProceduresTIOrdinal)

    })

});

}

logger.Information($"Templates: {count}");

}

```

```

// Result Set 4 - Mailboxes

```

```

if (reader.NextResult())

{

    var count = 0;

    var mailboxKeyMIOrdinal = reader.GetOrdinal("Mailbox_Key");

    var mailboxNameMIOrdinal = reader.GetOrdinal("Mailbox_Name");

    var mailboxCodeMIOrdinal = reader.GetOrdinal("Mailbox_Code");

    var destinationMailboxMIOrdinal = reader.GetOrdinal("Destination_Mailbox");

    var destinationCodeMIOrdinal = reader.GetOrdinal("Destination_Code");

    var destinationQualifierMIOrdinal = reader.GetOrdinal("Destination_Qualifier");

    var hondaPlantCodeMIOrdinal = reader.GetOrdinal("Honda_Plant_Code");

    var destinationInternalIdMIOrdinal = reader.GetOrdinal("Destination_Internal_Id");

    var elementSeperatorMIOrdinal = reader.GetOrdinal("Element_Separator");

    var subelementSeperatorMIOrdinal = reader.GetOrdinal("Subelement_Separator");

    var segmentTerminatorMIOrdinal = reader.GetOrdinal("Segment_Terminator");

    var mailboxPCNMIOrdinal = reader.GetOrdinal("Mailbox_PCN");

    var mailboxStatusMIOrdinal = reader.GetOrdinal("Mailbox_Status");

    var mailboxAddDateMIOrdinal = reader.GetOrdinal("Mailbox_Add_Date");

    var mailboxUpdateDateMIOrdinal = reader.GetOrdinal("Mailbox_Update_Date");

```

```
var vanConnectionNameMIOrdinal = reader.GetOrdinal("VAN_Connection_Name");
```

```
var valueAddedNetworkMIOrdinal = reader.GetOrdinal("Value_Added_Network");
```

```
while (reader.Read())
```

```
{
```

```
    count++;
```

```
    mailboxes.Add(new EDIUsageDataModel
```

```
    {
```

```
        TransactionDate = dateToLoad,
```

```
        JsonMessage = JsonConvert.SerializeObject(new EDIUsageModel
```

```
        {
```

```
            TransactionDate = dateToLoad,
```

```
            Type = MailboxInfo,
```

```
            MailboxKey = reader.GetInt32(mailboxKeyMIOrdinal),
```

```
            MailboxName = reader.GetString(mailboxNameMIOrdinal),
```

```
            MailboxCode = reader.GetString(mailboxCodeMIOrdinal),
```

```
            DestinationMailbox = reader.GetString(destinationMailboxMIOrdinal),
```

```
            DestinationCode = reader.GetString(destinationCodeMIOrdinal),
```

```
            DestinationQualifier = reader.GetString(destinationQualifierMIOrdinal),
```

```
            HondaPlantCode = reader.GetString(hondaPlantCodeMIOrdinal),
```

```
            DestinationInternalId = reader.GetString(destinationInternalIdMIOrdinal),
```

```
            ElementSeparator = reader.GetString(elementSeperatorMIOrdinal),
```

```
            SubelementSeparator = reader.GetString(subelementSeperatorMIOrdinal),
```

```
            SegmentTerminator = reader.GetInt32(segmentTerminatorMIOrdinal),
```

```
            MailboxPCN = reader.GetInt32(mailboxPCNMIOrdinal),
```

```
            MailboxStatus = reader.GetString(mailboxStatusMIOrdinal),
```

```

        MailboxAddDate = reader.GetString(mailboxAddDateMIOrdinal),

        MailboxUpdateDate = reader.GetString(mailboxUpdateDateMIOrdinal),

        VANConnectionName = reader.GetString(vanConnectionNameMIOrdinal),

        ValueAddedNetwork = reader.GetString(valueAddedNetworkMIOrdinal)

    })

});

}

logger.Information($"Mailboxes: {count}");
}

```

```

// Result Set 5 - Trading Partners

```

```

if (reader.NextResult())
{
    var count = 0;

    var tradingPartnerKeyTPIOrdinal = reader.GetOrdinal("Trading_Partner_Key");

    var tradingPartnerNameTPIOrdinal = reader.GetOrdinal("Trading_Partner");

    while (reader.Read())
    {
        count++;

        tradingPartners.Add(new EDIUsageDataModel
        {
            TransactionDate = dateToLoad,

            JsonMessage = JsonConvert.SerializeObject(new EDIUsageModel
            {
                TransactionDate = dateToLoad,

```

```

        Type = TradingPartnerInfo,

        TradingPartnerKey = reader.GetInt32(tradingPartnerKeyTPIOrdinal),

        TradingPartnerName = reader.GetString(tradingParterNameTPIOrdinal)

    })

});

}

logger.Information($"Trading Partners: {count}");

}

}

}

}

}

}

/// <summary>
/// This method is not yet used
/// </summary>
/// <param name="startDate"></param>
/// <param name="endDate"></param>
/// <param name="cancellationToken"></param>
/// <param name="logger"></param>
/// <returns></returns>

public Task<IEnumerable<EDIUsageDataModel>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
cancellationToken cancellationToken, IEtlProcessLogger logger)

{

    throw new NotImplementedException();

}
```

```
}  
  
}
```

Transformer.cs (Monitoring.ETL.Domain\EDIUsage\RabbitMQLoader\Transformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.EDIUsage.RabbitMQLoader
```

```
{
```

```
    public class Transformer : ITransformer<EDIUsageDataModel, RabbitMQ.Model.RabbitMQEDIUsageModel>
```

```
    {
```

```
        public async Task<IEnumerable<RabbitMQ.Model.RabbitMQEDIUsageModel>>
```

```
        TransformAsync(IEnumerable<EDIUsageDataModel> extracted, CancellationToken cancellationToken,
```

```
        IEtlProcessLogger logger)
```

```
        {
```

```
            await Task.Yield();
```

```
            return extracted.Select(pc =>
```

```
                new RabbitMQ.Model.RabbitMQEDIUsageModel
```

```

{

    JsonMessage = pc.JsonMessage

});

}

}

}

```

DelayFactory.cs (Monitoring.ETL.Domain\EDIUsage\WarehouseLoader\DelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.EDIUsage.RabbitMQExtractor
```

```

{

    /// <summary>

    /// Defines how often the the data load runs. If the total number of records loaded is beyond the

    /// MaxThresholdForDelay, then no delay occurs and the load process will immediately be executed again.

    ///

    /// If the number of records is less than the Threshold, then the process will sleep for DelayTimeSpan amount of time

    ///

    /// The data Extractor process should load data in chunks of records for a specific date range.

    /// </summary>

    public sealed class DelayFactory : ThresholdExtractionDelayFactory<RabbitMQ.Model.RabbitMQEDIUsageModel>

    {

        /// <summary>

        /// Time to delay when the threshold count is not reached

        /// </summary>

```

```
protected override TimeSpan DelayTimeSpan => new TimeSpan(0, 0, 10);
```

```
/// <summary>
```

```
/// number of records to reach before the delay is ignored.
```

```
/// </summary>
```

```
protected override int MaxThresholdForDelay => int.MaxValue;
```

```
}
```

```
}
```

EDIUsageWarehouseLoader.cs

(Monitoring.ETL.Domain\EDIUsage\WarehouseLoader\EDIUsageWarehouseLoader.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.EDIUsage.WarehouseLoader
```

```
{
```

```
    public class EDIUsageWarehouseLoader : Loader<Warehouse.Model.EDI_Usage_w>
```

```
    {
```

```
        public EDIUsageWarehouseLoader() : base("Fact_EDI_Usage_Add")
```

```
        {
```

```
        }
```

```
    }
```

```
}
```

Extractor.cs (Monitoring.ETL.Domain\EDIUsage\WarehouseLoader\Extractor.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.EDIUsage.RabbitMQExtractor
```

```
{
```

```
    public sealed class Extractor : IExtractor<RabbitMQ.Model.RabbitMQEDIUsageModel>
```

```
    {
```

```
        private RabbitMQ.EDIUsage.Consumer _consumer;
```

```
        public async Task<ResultSet<RabbitMQ.Model.RabbitMQEDIUsageModel>>
```

```
ExtractAllSinceLastExtractAsync(Cancellation_token cancellationToken, IEtlProcessLogger logger)
```

```
    {
```

```
        _consumer = _consumer ?? new RabbitMQ.EDIUsage.Consumer(logger);
```

```
        return await _consumer.ExtractAsync();
```

```
    }
```

```
        public Task<IEnumerable<RabbitMQ.Model.RabbitMQEDIUsageModel>> ExtractBetweenAsync(DateTime startDate,
```

```
DateTime endDate, Cancellation_token cancellationToken, IEtlProcessLogger logger)
```

```
    {
```

```
        throw new NotImplementedException();
```

```
    }
```

```
}
```



```
}
```

Transformer.cs (Monitoring.ETL.Domain\EDIUsage\WarehouseLoader\Transformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.EDIUsage.RabbitMQExtractor
```

```
{
```

```
    /// <summary>
```

```
    /// Load the messages from RabbitMQ into the table Template_Worktable.
```

```
    /// </summary>
```

```
    public class Transformer : ITransformer<RabbitMQ.Model.RabbitMQEDIUsageModel,
```

```
    Warehouse.Model.EDI_Usage_w>
```

```
    {
```

```
        public async Task<IEnumerable<Warehouse.Model.EDI_Usage_w>>
```

```
        TransformAsync(IEnumerable<RabbitMQ.Model.RabbitMQEDIUsageModel> extracted, CancellationToken
```

```
        cancellation_token, IEtlProcessLogger logger)
```

```
    {
```

```
        await Task.Yield();
```

```

return extracted.Select(pc => new Warehouse.Model.EDI_Usage_w

{

    JSON_Message = pc.JsonMessage

});

}

}

}

```

ElasticRepository.cs (Monitoring.ETL.Domain\ElasticSearch\ElasticRepository.cs):

```

using System;

using System.IO;

using System.Net;

using System.Net.Security;

using System.Text;

using System.Threading.Tasks;

using Newtonsoft.Json;

using Plex.Infrastructure.ConfigSystems;

using ETL.Process;

namespace Monitoring.ETL.Domain.ElasticSearch

{

    public abstract class ElasticRepository

    {

        private readonly string baseUrl;

        private readonly string credentials;

        public ElasticRepository()

```

```

{

var registry = new PlexRegistrySystem();

var host = registry.GetString("CloudOps", "Monitoring", "SIEM", "Host", null);

var port = registry.GetString("CloudOps", "Monitoring", "SIEM", "Port", null);

var userName = registry.GetString("CloudOps", "Monitoring", "SIEM", "Username", null);

var password = registry.GetString("CloudOps", "Monitoring", "SIEM", "Password", null);


baseUrl = string.Format(

    "https://{0}:{1}",

    //"http://{0}:9200",

    host,

    port);


credentials = Convert

    .ToBase64String(Encoding

        .ASCII.GetBytes(string

            .Format(

                "{0}:{1}",

                userName,

                password))));


// This is to deal with invalid ssl certs.

// This can be removed once our wild card cert is figured out.

ServicePointManager.Expect100Continue = true;

ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;

//ServicePointManager.ServerCertificateValidationCallback += (sender, cert, chain, errors) => { return true; };

```

```
ServicePointManager.ServerCertificateValidationCallback = new RemoteCertificateValidationCallback((sender, cert,
chain, errors) => { return true; });

}
```

```
protected async Task<ResultSet<T>> ExtractAsync<T>(string index, string type, string payload)
```

```
where T : IElasticExtractModel
```

```
{

    var results = new ResultSet<T>();

    ElasticsearchResponse response = null;

    var url = string.Format(

        "{0}/{1}/_search",

        baseUrl,

        index);

    try

    {

        var request = (HttpWebRequest)WebRequest.Create(url);

        request.Headers.Add(HttpRequestHeader.Authorization, string.Format("Basic {0}", credentials));

        request.ContentType = "application/json";

        request.Method = WebRequestMethods.Http.Post;

        var streamWriter = new StreamWriter(request.GetRequestStream());

        streamWriter.Write(payload);

        streamWriter.Flush();

        streamWriter.Close();

    }
```

```
var jsonString = new StreamReader(request.GetResponse()).GetResponseStream()).ReadToEnd();
```

```
response = JsonConvert.DeserializeObject<ElasticSearchResponse>(jsonString);
```

```
}
```

```
catch (Exception ex)
```

```
{
```

```
ex.Data.Add("X-Elastic-Url", url);
```

```
ex.Data.Add("X-Elastic-Payload", payload);
```

```
results.Exceptions.Add(ex);
```

```
}
```

```
if (response?.hits?.hits != null)
```

```
{
```

```
foreach (var hit in response.hits.hits)
```

```
{
```

```
try
```

```
{
```

```
var errorModel = JsonConvert.DeserializeObject<ElasticSearchResult<T>>(hit);
```

```
var model = errorModel._source;
```

```
model.Elastic_Id = errorModel._id;
```

```
results.Results.Add(model);
```

```
}
```

```
catch (Exception ex)
```

```
{
```

```
var exception = new Exception("Error parsing json object", ex);
```

```
exception.Data.Add("json", hit);
```

```
        results.Exceptions.Add(exception);

    }

}

}

return results;

}

}

}
```

ElasticSearchHits.cs (Monitoring.ETL.Domain\ElasticSearch\ElasticSearchHits.cs):

```
using Newtonsoft.Json;
```

```
namespace Monitoring.ETL.Domain.ElasticSearch

{

    internal class ElasticSearchHits

    {

        [JsonConverter(typeof(JsonObjectStringConverter))]

        public string[] hits { get; set; }

    }

}
```

ElasticSearchResponse.cs (Monitoring.ETL.Domain\ElasticSearch\ElasticSearchResponse.cs):

```
namespace Monitoring.ETL.Domain.ElasticSearch

{

    internal class ElasticSearchResponse
```

```
{  
  
    public int took { get; set; }  
  
    public bool timed_out { get; set; }  
  
    public ElasticSearchHits hits { get; set; }  
  
}  
  
}
```

ElasticSearchResult.cs (Monitoring.ETL.Domain\ElasticSearch\ElasticSearchResult.cs):

```
namespace Monitoring.ETL.Domain.ElasticSearch
```

```
{  
  
    internal class ElasticSearchResult<T>  
  
    {  
  
        public string _index { get; set; }  
  
        public string _type { get; set; }  
  
        public string _id { get; set; }  
  
        public T _source { get; set; }  
  
    }  
  
}
```

Error2Repository.cs (Monitoring.ETL.Domain\ElasticSearch>Error2Repository.cs):

```
using System;  
  
using System.Globalization;  
  
using System.Threading.Tasks;  
  
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.ElasticSearch
```

```

{

public class Error2Repository : ElasticRepository

{

public async Task<ResultSet<ErrorModel>> GetAllAfterKeyAsync(int errorKey, int maxCount)

{

var result = await ExtractAsync<ErrorModel>(

    "logs-pmc-app-*",

    null,

    string.Format(

        //{"size":{1},"sort":[{"error_key":{"order":"asc"}]},"query":{"range":{"error_key":{"gt":{0}}}}}}",

        @"{{

""size"": {1},

""sort"": [ {{ ""error_key"": {""order"":""asc"" } } } ],

""query"": {{

""bool"": {{

""filter"": [

{{ ""term"": {{ ""event_queue"": ""error2"" } } },

{{ ""range"": {{ ""error_key"": {""gt"":{0} } } } }

]

}}

}}

}},

    errorKey,

    maxCount));

return result;

```



```
}
```

```
public async Task<ResultSet<ErrorModel>> GetAllAfterKeyAsync(DateTime date, int maxCount)
```

```
{
```

```
    var result = await ExtractAsync<ErrorModel>(  
        "logs-pmc-app-*",  
        null,  
        string.Format(  
            @"{{  
                ""size"": {1},  
                ""sort"": [ {{ ""@timestamp"": {{ ""order"": ""asc"" }} } ],  
                ""query"": {{  
                    ""bool"": {{  
                        ""filter"": [  
                            {{ ""range"": {{ ""@timestamp"": {{ ""gt"": ""{0}"" , ""format"": ""yyyy-MM-dd HH:mm:ss.SSS"" }} }} }  
                        ]  
                    }}  
                }}  
            } }",  
            date.ToUniversalTime().ToString("yyyy-MM-dd HH:mm:ss.fff", CultureInfo.InvariantCulture),  
            maxCount));  
  
    return result;  
}
```

```
}
```

```
}
```

```
}
```

ErrorExtractor.cs (Monitoring.ETL.Domain\ElasticSearch>ErrorExtractor.cs):

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.ElasticSearch

{

public class ErrorExtractor : IExtractor<ErrorModel>

{

private readonly Error2Repository _errorRepository;

private readonly ErrorState _state;

public ErrorExtractor()

{

_errorRepository = new Error2Repository();

_state = new ErrorState();

}

public async Task<ResultSet<ErrorModel>> ExtractAllSinceLastExtractAsync(Cancellation token cancellation token,

IEtlProcessLogger logger)

{

try

```

{

    var lastDate = _state.GetLastDate();

    var errors = await _errorRepository.GetAllAfterKeyAsync(lastDate, 5000);

    if (errors.Results.Any())
    {
        await _state.SetLastKey(errors.Results.Select(e => e.Error_Key).OrderBy(k => k).Last());

        await _state.SetLastDate(errors.Results.Select(e => e.Timestamp).OrderBy(k => k).Last().ToLocalTime());
    }

    return errors;

}

catch (Exception e)

{

    var results = new ResultSet<ErrorModel>();

    results.Exceptions.Add(e);

    return results;

}

}

public Task<IEnumerable<ErrorModel>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
CancellationTokens cancellationTokens, IEtlProcessLogger logger)

{

    throw new NotImplementedException();

}

```

```
}
```

```
}
```

ErrorModel.cs (Monitoring.ETL.Domain\ElasticSearch>ErrorModel.cs):

```
using System.Collections.Generic;
```

```
namespace Monitoring.ETL.Domain.ElasticSearch
```

```
{
```

```
    public class ErrorModel : SiemModel
```

```
    {
```

```
        public int Error_Key { get; set; }
```

```
        public string Filename { get; set; }
```

```
        public string Path_Info { get; set; }
```

```
        public string Script_Name { get; set; }
```

```
        public string Lock_Chain { get; set; }
```

```
        public string Sql_Server { get; set; }
```

```
        public string Path_Translated { get; set; }
```

```
        public string Element_List { get; set; }
```

```
        public string Server_Name { get; set; }
```

```
        public string Session_Id { get; set; }
```

```
        public string Session_Info { get; set; }
```

```
        public string Setting_Group { get; set; }
```

```
        public string Setting_Name { get; set; }
```

```
        public string Exception_Data_Source_Key { get; set; }
```

```
        public string Exception_Data_Source_Name { get; set; }
```

```
        public string Exception_Column_Name { get; set; }
```

```

    public string Exception_SQL_Command { get; set; }

    public string Exception_Error_Message { get; set; }

    public string Missing_Image_Path { get; set; }

    public string Stack_Trace { get; set; }

    public string Exception_Location { get; set; }

    public string ApplicationName { get; set; }

    public string Referer_Path { get; set; }

}

}

```

ErrorState.cs (Monitoring.ETL.Domain\ElasticSearch\ErrorState.cs):

```

using System;

using System.Threading.Tasks;


namespace Monitoring.ETL.Domain.ElasticSearch
{
    public sealed class ErrorState
    {
        private readonly LoadStateRepository _repository = new LoadStateRepository();

        private int? _lastKey;

        private DateTime? _lastDate;


        public int GetLastKey()
        {
            if (_lastKey.HasValue == false)
            {

```

```
        _lastKey = _repository.GetKeyFor(LoadName.Error);
    }

    return _lastKey.HasValue ? _lastKey.Value : 0;
}

public DateTime GetLastDate()
{
    if (_lastDate.HasValue == false)
    {
        _lastDate = _repository.GetDateFor(LoadName.Error);
    }

    return _lastDate.HasValue ? _lastDate.Value : DateTime.MinValue;
}

public async Task SetLastKey(int key)
{
    _lastKey = key;

    await _repository.SetStateAsync(LoadName.Error, key);
}

public async Task SetLastDate(DateTime date)
{
    _lastDate = date;
```

```
        await _repository.SetStateAsync(LoadName.Error, date);

    }

}

}
```

IElasticExtractModel.cs (Monitoring.ETL.Domain\ElasticSearch\IElasticExtractModel.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.ElasticSearch

{

    public interface IElasticExtractModel : IExtractModel

    {

        string Elastic_Id { get; set; }

    }

}
```

ILoadStateRepository.cs (Monitoring.ETL.Domain\ElasticSearch\ILoadStateRepository.cs):

```
using System;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.ElasticSearch

{

    public interface ILoadStateRepository

    {

        DateTime GetDate();

    }

}
```

```
int GetKey();

Task SetDate(DateTime date);

Task SetKey(int key);

}

}
```

JsonObjectStringConverter.cs (Monitoring.ETL.Domain\ElasticSearch\JsonObjectStringConverter.cs):

```
using System;

using System.Linq;

using Newtonsoft.Json;

using Newtonsoft.Json.Linq;

namespace Monitoring.ETL.Domain.ElasticSearch
{
    public class JsonObjectStringConverter : JsonConverter
    {
        public override bool CanConvert(Type objectType)
        {
            return objectType == typeof(string) || objectType == typeof(string[]);
        }

        public override object ReadJson(JsonReader reader, Type objectType, object existingValue, JsonSerializer serializer)
        {
            JToken token = JToken.Load(reader);

            if (token.Type == JTokenType.Array)
            {

```



```

        return token.Children().Select(c => c.ToString()).ToArray();
    }

    return null;
}

public override void WriteJson(JsonWriter writer, object value, JsonSerializer serializer)
{
    throw new NotImplementedException();
}
}
}
}

```

LoadName.cs (Monitoring.ETL.Domain\ElasticSearch\LoadName.cs):

```

namespace Monitoring.ETL.Domain.ElasticSearch
{
    public enum LoadName
    {
        Error = 1
    }
}

```

LoadStateRepository.cs (Monitoring.ETL.Domain\ElasticSearch\LoadStateRepository.cs):

```

using System;

using System.Data.Entity;

using System.Linq;

```

```
using System.Threading.Tasks;
```

```
using Monitoring.ETL.Domain.Model;
```

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.ElasticSearch
```

```
{  
  
    public class LoadStateRepository<T> : ILoadStateRepository  
    {  
  
        private readonly string _className;  
  
        private readonly LoadStateRepository _repository = new LoadStateRepository();  
  
        public LoadStateRepository()  
        {  
            _className = typeof(T).Name;  
        }  
  
        public LoadStateRepository(string serverId)  
        {  
            _className = string.Format("{0}:{1}", typeof(T).Name, serverId);  
        }  
  
        public DateTime GetDate()  
        {  
            return _repository.GetDateFor(_className);  
        }  
    }  
}
```

```
public int GetKey()

{

    return _repository.GetKeyFor(_className);

}
```

```
public async Task SetDate(DateTime date)

{

    await _repository.SetStateAsync(_className, date);

}
```

```
public async Task SetKey(int key)

{

    await _repository.SetStateAsync(_className, key);

}

}
```

```
public sealed class LoadStateRepository

{

    private readonly DbContext _warehouse;


    public LoadStateRepository()

        : this(new WarehouseServerProvider())

    {

    }

}
```

```
internal LoadStateRepository(IWarehouseServerProvider warehouseServerProvider)

{

    var server = warehouseServerProvider.GetWarehouseServerName();

    _warehouse = new Meta_WarehouseEntities().ChangeDatabase(dataSource: server);

}
```

```
public long GetBigIntKeyFor(string loadName)

{

    var task = GetFor(loadName);

    task.Wait();

    return task.Result.Last_Load_Bigint_Key ?? 0;

}
```

```
public long GetBigIntKeyFor(LoadName loadName)

{

    return GetBigIntKeyFor(loadName.ToString());

}
```

```
public string GetConnection()

{

    return _warehouse.Database.Connection.ConnectionString;

}
```

```
public DateTime GetDateFor(string loadName)

{

    var task = GetFor(loadName);
```

```
task.Wait();

return task.Result.Last_Load_Date ?? DateTime.MinValue;

}
```

```
public DateTime GetDateFor(LoadName loadName)

{

    return GetDateFor(loadName.ToString());

}
```

```
public int GetKeyFor(string loadName)

{

    var task = GetFor(loadName);

    task.Wait();

    return task.Result.Last_Load_Key ?? 0;

}
```

```
public int GetKeyFor(LoadName loadName)

{

    return GetKeyFor(loadName.ToString());

}
```

```
public async Task SetStateAsync(LoadName loadName, int key)

{

    await SetStateAsync(loadName.ToString(), key);

}
```

```
public async Task SetStateAsync(string loadName, int key)

{

    var state = await GetFor(loadName);

    if (state.Last_Load_Key.HasValue == false || key > state.Last_Load_Key)

    {

        state.Last_Load_Key = key;

        await _warehouse.SaveChangesAsync();

    }

}
```

```
public async Task SetStateAsync(LoadName loadName, long key)

{

    await SetStateAsync(loadName.ToString(), key);

}
```

```
public async Task SetStateAsync(string loadName, long key)

{

    var state = await GetFor(loadName);

    if (state.Last_Load_Bigint_Key.HasValue == false || key > state.Last_Load_Bigint_Key)

    {

        state.Last_Load_Bigint_Key = key;

        await _warehouse.SaveChangesAsync();

    }

}
```

```
public async Task SetStateAsync(LoadName loadName, DateTime date)
```

```
{  
  
    await SetStateAsync(loadName.ToString(), date);  
  
}
```

```
public async Task SetStateAsync(string loadName, DateTime date)  
  
{  
  
    var state = await GetFor(loadName);  
  
    if (!state.Last_Load_Date.HasValue || date > state.Last_Load_Date)  
  
    {  
  
        state.Last_Load_Date = date;  
  
        await _warehouse.SaveChangesAsync();  
  
    }  
  
}
```

```
private async Task<Load_State> GetFor(string loadName)  
  
{  
  
    var loadStateSet = _warehouse.Set<Load_State>();  
  
  
  
    var value = loadStateSet.SingleOrDefault(s => s.Load_Name == loadName);  
  
  
  
    if (value == null)  
  
    {  
  
        value = loadStateSet.Create();  
  
  
  
        value.Load_Name = loadName;  
  

```

```

        loadStateSet.Add(value);

        await _warehouse.SaveChangesAsync();
    }

    return value;
}
}
}

```

SiemModel.cs (Monitoring.ETL.Domain\ElasticSearch\SiemModel.cs):

```

using System;

using Newtonsoft.Json;

namespace Monitoring.ETL.Domain.ElasticSearch
{
    public abstract class SiemModel : IElasticExtractModel
    {
        public string Elastic_Id { get; set; }

        [JsonProperty("@timestamp")]
        public DateTime Timestamp { get; set; }

        public string Message { get; set; }

        public string Event_Division { get; set; }

        public string Event_Source { get; set; }

        public string Event_Queue { get; set; }

        public string Severity { get; set; }
    }
}

```



```
public string Environment { get; set; }

[JsonProperty("@version")]

public string Version { get; set; }

public string Event_Type { get; set; }

[JsonConverter(typeof(StringArrayConverter))]

public string[] Event_Sub_Type { get; set; }

public string[] Event_Sub_Type_Matches { get; set; }

public int PUN { get; set; }

public int PCN { get; set; }

public string Destination_Hostname { get; set; }

public string Referer { get; set; }

public string Source_IP { get; set; }

public string Source_Hostname { get; set; }

public string Method { get; set; }

public string User_Agent { get; set; }

public string User_Agent_Device { get; set; }

public string User_Agent_Major { get; set; }

public string User_Agent_Name { get; set; }

public string User_Agent_OS { get; set; }

public string User_Agent_Patch { get; set; }

public string Data_Center { get; set; }

[JsonConverter(typeof(StringArrayConverter))]

public string[] Tags { get; set; }

}

}
```

StringArrayConverter.cs (Monitoring.ETL.Domain\ElasticSearch\StringArrayConverter.cs):

```
using System;
```

```
using System.Linq;
```

```
using Newtonsoft.Json;
```

```
using Newtonsoft.Json.Linq;
```

```
namespace Monitoring.ETL.Domain.ElasticSearch
```

```
{
```

```
    public class StringArrayConverter : JsonConverter
```

```
    {
```

```
        public override bool CanConvert(Type objectType)
```

```
        {
```

```
            return objectType == typeof(string) || objectType == typeof(string[]);
```

```
        }
```

```
        public override object ReadJson(JsonReader reader, Type objectType, object existingValue, JsonSerializer serializer)
```

```
        {
```

```
            JToken token = JToken.Load(reader);
```

```
            if (token.Type == JTokenType.String)
```

```
            {
```

```
                return new string[] { token.ToString() };
```

```
            }
```

```
            else if (token.Type == JTokenType.Array)
```

```
            {
```

```
                return token.Children().Select(c => c.ToString()).ToArray();
```

```
            }
```

```

        return new string[0];
    }

    public override void WriteJson(JsonWriter writer, object value, JsonSerializer serializer)
    {
        throw new NotImplementedException();
    }
}
}
}

```

ErrorExtractionDelayFactory.cs (Monitoring.ETL.Domain\ErrorExtractionDelayFactory.cs):

```

using System;

using System.Threading;

using System.Threading.Tasks;


using ETL.Process;


using Monitoring.ETL.Domain.ElasticSearch;

using Monitoring.ETL.Process;


namespace Monitoring.ETL.Domain
{
    public class ErrorExtractionDelayFactory : IExtractionDelayFactory<ErrorModel>
    {
        public async Task Create(ResultSet<ErrorModel> extracted, CancellationToken cancellationToken,

```

IEtlProcessLogger logger)

```
{  
    if (extracted == null || extracted.Results.Count + extracted.Exceptions.Count < 500)  
    {  
        await Task.Delay(TimeSpan.FromSeconds(10), cancellationToken);  
    }  
}  
}  
}
```

EtlSettings.cs (Monitoring.ETL.Domain\EtlSettings.cs):

using System;

using System.Configuration;

using Monitoring.UserExtensions;

namespace Monitoring.ETL.Domain

```
{  
    /// <summary>  
    /// Configurable settings from the app.config file  
    /// </summary>  
    public static class EtlSettings  
    {  
        /// <summary>  
        /// AppSetting = serverId  
        /// The N scale server ID to process
```

```
/// </summary>
```

```
/// <remarks>
```

```
/// If this value is retrieved, it MUST be set in your application config,
```

```
/// otherwise a null reference exception will be thrown
```

```
/// </remarks>
```

```
public static string ServerId
```

```
{
```

```
    get
```

```
{
```

```
    var serverId = ConfigurationManager.AppSettings["serverId"];
```

```
    if (string.IsNullOrEmpty(serverId))
```

```
{
```

```
        throw new NullReferenceException("ServerId not set in the application.config file");
```

```
}
```

```
    return serverId;
```

```
}
```

```
}
```

```
/// <summary>
```

```
/// The region of users to pull for Plexus_User ETL
```

```
/// </summary>
```

```
public static string UserRegion
```

```
{
```

```
    get
```

```
{
```

```
    var userRegion = ConfigurationManager.AppSettings["UserRegion"];
```

```
        if (string.IsNullOrEmpty(userRegion))
        {
            throw new NullReferenceException("UserRegion not set in the application.config file. This key need to
match Dim_Data_Center.Data_Center_Abbreviation");
        }

        return userRegion;
    }
}
```

```
public static int DataCenterKey
{
    get
    {
        var dataCenterKey = ConfigurationManager.AppSettings["DataCenterKey"].ToInt();

        if (dataCenterKey == 0)
        {
            throw new Exception("DataCenterKey not defined in .config file");
        }

        return dataCenterKey;
    }
}

/// <summary>
/// AppSettings = SprocCommandTimeout
/// Timeout for the sproc timeout
```

```
/// </summary>
```

```
/// <remarks>
```

```
/// Defaults to 300 seconds if not set</remarks>
```

```
public static int SprocCommandTimeout
```

```
{
```

```
    get
```

```
    {
```

```
        if (!int.TryParse(ConfigurationManager.AppSettings["SprocCommandTimeout"], out int v))
```

```
            v = 300;
```

```
        return v;
```

```
    }
```

```
}
```

```
/// <summary>
```

```
/// AppSettings = BatchSize
```

```
/// The batch size to process
```

```
/// </summary>
```

```
/// <remarks>
```

```
/// Defaults to 20000 if not set</remarks>
```

```
public static ushort BatchSize(ushort defaultBatchSize = 20000)
```

```
{
```

```
    if (!ushort.TryParse(ConfigurationManager.AppSettings["BatchSize"], out ushort batchSize))
```

```
        batchSize = defaultBatchSize;
```

```
    return batchSize;
```

```
}
```

```
/// <summary>
```

```
/// AppSettings = ThresholdCount
```

```
/// </summary>
```

```
/// <param name="defaultValue"></param>
```

```
/// <returns></returns>
```

```
public static int ThresholdCount(int defaultValue = 100)
```

```
{
```

```
    if (!int.TryParse(ConfigurationManager.AppSettings["ThresholdCount"], out int value))
```

```
        value = defaultValue;
```

```
    return value;
```

```
}
```

```
/// <summary>
```

```
/// The sentryId instance ID
```

```
/// </summary>
```

```
public static string SentryId => ConfigurationManager.AppSettings["sentryId"];
```

```
/// <summary>
```

```
/// The
```

```
/// </summary>
```

```
public static string Environment => ConfigurationManager.AppSettings["environment"];
```

```
public static int ThresholdDelayHours(int defaultValue = 0)
```

```
{
```

```
    if (!int.TryParse(ConfigurationManager.AppSettings["ThresholdDelayHours"], out int value))
```



```

        value = defaultValue;

    return value;

}

public static int ThresholdDelayMinutes(int defaultValue = 0)

{

    if (!int.TryParse(ConfigurationManager.AppSettings["ThresholdDelayMinutes"], out int value))

        value = defaultValue;

    return value;

}

public static int ThresholdDelaySeconds(int defaultValue = 0)

{

    if (!int.TryParse(ConfigurationManager.AppSettings["ThresholdDelaySeconds"], out int value))

        value = defaultValue;

    return value;

}

}

}

```

ExceptionExtensions.cs (Monitoring.ETL.Domain\ExceptionExtensions.cs):

```

using System;

using System.Data.SqlClient;

namespace Monitoring.ETL.Domain

{

```

```

public static class ExceptionExtensions
{
    public static bool IsNetworkConnectionException(this Exception ex)
    {
        if (ex == null)
            return false;

        if (ex is SqlException sqlException)
        {
            foreach (SqlError e in sqlException.Errors)
            {
                if (e.Number == 1225)
                {
                    return true;
                }
            }
        }

        return false;
    }
}

```

RabbitMQExtractionDelayFactory.cs

(Monitoring.ETL.Domain\Fact\DatabasePerformance\RabbitMQExtractionDelayFactory.cs):

```

using System;

```

```
namespace Monitoring.ETL.Domain.Fact.DatabasePerformance
```

```
{  
  
    public class RabbitMQExtractionDelayFactory :  
ThresholdExtractionDelayFactory<RabbitMQ.Model.DatabasePerformanceMeasurement>  
  
    {  
  
        protected override int MaxThresholdForDelay => 1000;  
  
        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);  
  
    }  
}
```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\Fact\DatabasePerformance\RabbitMQExtractor.cs):

```
using System;  
  
using System.Collections.Generic;  
  
using System.Threading;  
  
using System.Threading.Tasks;  
  
  
using ETL.Process;  
  
  
using Monitoring.ETL.Process;  
  
  
namespace Monitoring.ETL.Domain.Fact.DatabasePerformance  
  
{  
  
    public sealed class RabbitMQExtractor : IExtractor<RabbitMQ.Model.DatabasePerformanceMeasurement>  
  
    {
```

```
private RabbitMQ.DatabasePerformanceMeasurement.Consumer _consumer;
```

```
public async Task<ResultSet<RabbitMQ.Model.DatabasePerformanceMeasurement>>
```

```
ExtractAllSinceLastExtractAsync(Cancellation_token cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    _consumer = _consumer ?? new RabbitMQ.DatabasePerformanceMeasurement.Consumer(logger);  
  
    return await _consumer.ExtractAsync();  
  
}
```

```
public Task<IEnumerable<RabbitMQ.Model.DatabasePerformanceMeasurement>>
```

```
ExtractBetweenAsync(DateTime startDate, DateTime endDate, Cancellation_token cancellationToken,  
IEtlProcessLogger logger)
```

```
{  
  
    throw new NotImplementedException();  
  
}  
  
}  
  
}
```

RabbitMQLoader.cs (Monitoring.ETL.Domain\Fact\DatabasePerformance\RabbitMQLoader.cs):

```
namespace Monitoring.ETL.Domain.Fact.DatabasePerformance
```

```
{  
  
    public class RabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.DatabasePerformanceMeasurement>  
  
    {  
  
        public RabbitMQLoader() : base(new RabbitMQ.DatabasePerformanceMeasurement.Producer())  
  
        {  
  
        }  
  
    }  
  
}
```

```
}
```

```
}
```

RabbitMQWarehouseTransformer.cs

(Monitoring.ETL.Domain\Fact\DatabasePerformance\RabbitMQWarehouseTransformer.cs):

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Fact.DatabasePerformance
```

```
{
```

```
    public class RabbitMQWarehouseTransformer : ITransformer<RabbitMQ.Model.DatabasePerformanceMeasurement,  
Warehouse.Model.Performance_Analysis_Data_w>
```

```
    {
```

```
        public async Task<IEnumerable<Warehouse.Model.Performance_Analysis_Data_w>>  
TransformAsync(IEnumerable<RabbitMQ.Model.DatabasePerformanceMeasurement> extracted,  
Cancellation token cancellationToken, IETLProcessLogger logger)
```

```
    {
```

```
        await Task.Yield();
```

```
        return extracted;
```

```
    }
```

```
}
```

```
}
```

DelayFactory.cs (Monitoring.ETL.Domain\Fact\DatabasePerformance\Sentry\DelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Fact.DatabasePerformance.Sentry
```

```
{
```

```
    public class DelayFactory : ThresholdExtractionDelayFactory<PerformanceAnalysisData>
```

```
    {
```

```
        protected override int MaxThresholdForDelay => 100;
```

```
        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);
```

```
    }
```

```
}
```

Extractor.cs (Monitoring.ETL.Domain\Fact\DatabasePerformance\Sentry\Extractor.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.ElasticSearch;
```

```
using Monitoring.ETL.Process;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Data.SqlClient;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.Fact.DatabasePerformance.Sentry
```

```
{
```

```
    public class Extractor : IExtractor<PerformanceAnalysisData>
```

```
    {
```

```
        private readonly LoadStateRepository<PerformanceAnalysisData> _loadStateRepository;
```

```
        private readonly Repository _repository;
```

```
        private int _lastDateTimeStamp;
```

```
        public Extractor()
```

```
        {
```

```
            _loadStateRepository = new LoadStateRepository<PerformanceAnalysisData>();
```

```
            var serverConnectionIds = GetServers();
```

```
            Console.WriteLine($"Servers found: {serverConnectionIds.Count}");
```

```
            foreach (var c in serverConnectionIds)
```

```
            {
                Console.WriteLine(c.Instance_Name);
            }

```

```
            var counters = new List<PerformanceAnalysisCounter>
```

```
            {
```

```
                new PerformanceAnalysisCounter { Id = 182, Name = "Transactions/Sec", DatabaseLevel = true },
```

```
                new PerformanceAnalysisCounter { Id = 212, Name = "Batches/Sec" },
```

```
                new PerformanceAnalysisCounter { Id = 144, Name = "Page Reads/Sec" },
```

```
                new PerformanceAnalysisCounter { Id = 142, Name = "Page Life Expectancy" },
```

```

new PerformanceAnalysisCounter { Id = 143, Name = "Page Lookups/Sec" },

new PerformanceAnalysisCounter { Id = 1134, Name = "ASYNC_NETWORK_IO" },

new PerformanceAnalysisCounter { Id = 1135, Name = "BACKUP" },

new PerformanceAnalysisCounter { Id = 1139, Name = "BACKUPIO" },

new PerformanceAnalysisCounter { Id = 1172, Name = "CXPACKET" },

new PerformanceAnalysisCounter { Id = 1220, Name = "IO_COMPLETION" },

new PerformanceAnalysisCounter { Id = 1253, Name = "LOGBUFFER" },

new PerformanceAnalysisCounter { Id = 1272, Name = "PAGEIOLATCH_EX" },

new PerformanceAnalysisCounter { Id = 1275, Name = "PAGEIOLATCH_SH" },

new PerformanceAnalysisCounter { Id = 1276, Name = "PAGEIOLATCH_UP" },

new PerformanceAnalysisCounter { Id = 1278, Name = "PAGELATCH_EX" },

new PerformanceAnalysisCounter { Id = 1281, Name = "PAGELATCH_SH" },

new PerformanceAnalysisCounter { Id = 1282, Name = "PAGELATCH_UP" },

new PerformanceAnalysisCounter { Id = 1389, Name = "WRITELOG" }

};

_repository = new Repository(EtlSettings.SentryId, serverConnectionIds, counters);

}

public async Task<ResultSet<PerformanceAnalysisData>> ExtractAllSinceLastExtractAsync(CancellationTok
cancellationToken, IEtlProcessLogger logger)

{

    var resultSet = new ResultSet<PerformanceAnalysisData>();

    try

    {

```



```

if (_lastDateTimeStamp == default)
{
    _lastDateTimeStamp = _loadStateRepository.GetKey();
}

var measurements = await _repository.GetAllAfterDateTimeStamp(_lastDateTimeStamp - 60,
cancellationToken);

foreach (var measurement in measurements)
{
    resultSet.Results.Add(measurement);
}

var max = measurements
    .Select(d => d.End_Sentry_Timestamp)
    .OrderByDescending(date => date)
    .FirstOrDefault();

if (max > _lastDateTimeStamp)
{
    _lastDateTimeStamp = Convert.ToInt32(max);
    await _loadStateRepository.SetKey(_lastDateTimeStamp);
}
}

catch (Exception ex)
{
    resultSet.Exceptions.Add(ex);
}

```

```
}
```

```
return resultSet;
```

```
}
```

```
public Task<IEnumerable<PerformanceAnalysisData>> ExtractBetweenAsync(DateTime startDate, DateTime  
endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
{
```

```
    throw new NotImplementedException();
```

```
}
```

```
private T Get<T>(SqlDataReader reader, int ordinal)
```

```
{
```

```
    var value = reader.GetValue(ordinal);
```

```
    try
```

```
    {
```

```
        return (reader.IsDBNull(ordinal)) ? default : (T)reader.GetValue(ordinal);
```

```
    }
```

```
    catch (Exception)
```

```
    {
```

```
        return default;
```

```
    }
```

```
}
```

```
private List<ServerSourceConnection> GetServers()
```

```
{
```

```
var servers = new List<ServerSourceConnection>();
```

```
using (var connection = new ConsulConnectionFactory().Create("sqlperf", "Meta_Warehouse"))
```

```
using (var command = connection.CreateCommand())
```

```
{
```

```
    command.CommandText = "$@"
```

```
USE Meta_Warehouse;
```

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

```
SELECT
```

```
    I.Event_Source_Connection_Id,
```

```
    I.Machine_Name,
```

```
    I.Instance_Name
```

```
FROM dbo.Dim_Sentry_One_Event_Source_Connection_Id AS I
```

```
WHERE I.Effective = 1 and Dim_Data_Center_Key = {EtlSettings.DataCenterKey} ";
```

```
    connection.Open();
```

```
    var reader = command.ExecuteReader();
```

```
    if (reader.HasRows)
```

```
    {
```

```
        while (reader.Read())
```

```
        {
```

```
            servers.Add(new ServerSourceConnection
```

```
            {
```

```
                Id = Get<short>(reader, 0),
```

```

        Machine_Name = Get<string>(reader, 1),

        Instance_Name = Get<string>(reader, 2)

    });

}

}

else

{

    throw new Exception($"No entries found in Dim_Sentry_One_Event_Source_Connection_Id with
Effective=1 and Dim_Data_Center_Key={EtlSettings.DataCenterKey}");

}

    connection.Close();

}

return servers;

}

}

}

```

PerformanceAnalysisCounter.cs

(Monitoring.ETL.Domain\Fact\DatabasePerformance\Sentry\PerformanceAnalysisCounter.cs):

namespace Monitoring.ETL.Domain.Fact.DatabasePerformance.Sentry

```

{

    public class PerformanceAnalysisCounter

    {

        public string Name { get; set; }
    }
}

```

```

        public int Id { get; set; }

        public bool DatabaseLevel { get; set; }
    }

    public class ServerSourceConnection
    {
        public string Machine_Name { get; set; }

        public string Instance_Name { get; set; }

        public int Id { get; set; }
    }
}

```

PerformanceAnalysisData.cs

(Monitoring.ETL.Domain\Fact\DatabasePerformance\Sentry\PerformanceAnalysisData.cs):

```

using ETL.Process;

namespace Monitoring.ETL.Domain.Fact.DatabasePerformance.Sentry
{
    public class PerformanceAnalysisData : IExtractModel
    {
        public short Event_Source_Connection_Id { get; set; }

        public short Performance_Analysis_Counter_Id { get; set; }

        public long Start_Sentry_Timestamp { get; set; }

        public long End_Sentry_Timestamp { get; set; }

        public float Metric_Value { get; set; }
    }
}

```

```

        public bool Amendment_To_Totals { get; set; }

        public int Sample_Count { get; set; }

        public string Instance_Name { get; set; }

        public string Machine_Name { get; internal set; }

        public string Server_Instance_Name { get; set; }

    }
}

```

RabbitMQTransformer.cs (Monitoring.ETL.Domain\Fact\DatabasePerformance\Sentry\RabbitMQTransformer.cs):

```

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;


using ETL.Process;


using Monitoring.ETL.Process;


namespace Monitoring.ETL.Domain.Fact.DatabasePerformance.Sentry
{
    public class RabbitMQTransformer : ITransformer<PerformanceAnalysisData,
RabbitMQ.Model.DatabasePerformanceMeasurement>
    {
        public async Task<IEnumerable<RabbitMQ.Model.DatabasePerformanceMeasurement>>
TransformAsync(IEnumerable<PerformanceAnalysisData> extracted, CancellationToken cancellationToken,

```

IEtlProcessLogger logger)

```
{

    await Task.Yield();

    var dataCenterKey= EtlSettings.DataCenterKey;

    if (dataCenterKey == -1)

        throw new Exception("DataCenterKey not set in app.config. This must be set to match Dim_Data_Center");

    return extracted.Select(pad => new RabbitMQ.Model.DatabasePerformanceMeasurement

    {

        Event_Source_Connection_Id = pad.Event_Source_Connection_Id,

        Performance_Analysis_Counter_Id = pad.Performance_Analysis_Counter_Id,

        Start_Sentry_Timestamp = Convert.ToInt32(pad.Start_Sentry_Timestamp),

        End_Sentry_Timestamp = Convert.ToInt32(pad.End_Sentry_Timestamp),

        Amendment_To_Totals = pad.Amendment_To_Totals,

        Metric_Value = pad.Metric_Value,

        Sample_Count = pad.Sample_Count,

        Instance_Name = pad.Instance_Name,

        Machine_Name = pad.Machine_Name,

        Server_Instance_Name = pad.Server_Instance_Name,

        Data_Center_Key = dataCenterKey

    });

}

}
```

Repository.cs (Monitoring.ETL.Domain\Fact\DatabasePerformance\Sentry\Repository.cs):

using Monitoring.ETL.Domain.Warehouse;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

namespace Monitoring.ETL.Domain.Fact.DatabasePerformance.Sentry

{

internal class Repository

{

private readonly ConsulConnectionFactory _connectionFactory;

private readonly string _consulId;

private readonly string _database = "SENTRYONE";

private readonly string _sql;

private readonly string _intervalSql;

private readonly Dictionary<int, ServerSourceConnection> _servers;

public Repository(

string consulId,

List<ServerSourceConnection> serverConnections,

List<PerformanceAnalysisCounter> counters)


```

{

    _connectionFactory = new ConsulConnectionFactory();

    _consulId = consulId;

    _servers = serverConnections.ToDictionary(s => s.Id);


    _intervalSql = string.Format(@"

USE {0};

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;


SELECT

    C.Id,

    I.IntervalInTicks / 10000000 / 5

FROM dbo.PerformanceAnalysisCounter AS C

JOIN dbo.PerformanceAnalysisSampleInterval AS I

    ON I.ID = C.PerformanceAnalysisSampleIntervalID

WHERE C.Id IN ({1});",

        _database,

        string.Join(",", counters.Select(c => c.Id))

    );

    _sql = string.Format(@"

USE {0};

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;


DECLARE @End_Date_Time_Stamp INT;

```

SELECT

@End_Date_Time_Stamp = P.Timestamp

FROM dbo.PerformanceAnalysisDataDatabaseCounter AS P

WHERE P.[Timestamp] > @Last_Date_Time_Stamp

AND P.InstanceName != '_Total'

AND P.PerformanceAnalysisCounterID IN ({3})

AND P.EventSourceConnectionId IN ({1})

ORDER BY

P.[Timestamp]

OFFSET @Count - 1 ROWS

FETCH NEXT 1 ROWS ONLY;

SELECT

P.EventSourceConnectionID AS Event_Source_Connection_Id,

P.PerformanceAnalysisCounterID AS Performance_Analysis_Counter_Id,

NULL AS Instance_Name,

P.[Timestamp] AS End_Sentry_Timestamp,

AVG(P.[Value]) AS Metric_Value,

COUNT(*) AS Sample_Count

FROM dbo.PerformanceAnalysisData AS P

WHERE P.[Timestamp] > @Last_Date_Time_Stamp

AND (@End_Date_Time_Stamp IS NULL OR P.[Timestamp] <= @End_Date_Time_Stamp)

AND P.PerformanceAnalysisCounterID IN ({2})

AND P.EventSourceConnectionId IN ({1})

GROUP BY

P.[Timestamp],

P.PerformanceAnalysisCounterID,

P.EventSourceConnectionID

UNION ALL

SELECT

P.EventSourceConnectionID AS Event_Source_Connection_Id,

P.PerformanceAnalysisCounterID AS Performance_Analysis_Counter_Id,

P.InstanceName AS Instance_Name,

P.[Timestamp] AS End_Sentry_Timestamp,

AVG(P.[Value]) AS Metric_Value,

COUNT(*) AS Sample_Count

FROM dbo.PerformanceAnalysisDataDatabaseCounter AS P

WHERE P.[Timestamp] > @Last_Date_Time_Stamp

AND (@End_Date_Time_Stamp IS NULL OR P.[Timestamp] <= @End_Date_Time_Stamp)

AND P.InstanceName != '_Total'

AND P.PerformanceAnalysisCounterID IN ({3})

AND P.EventSourceConnectionId IN ({1})

GROUP BY

P.[Timestamp],

P.PerformanceAnalysisCounterID,

P.EventSourceConnectionID,

P.InstanceName",

_database,

string.Join(",", serverConnections.Select(s => s.Id)),

string.Join(",", counters.Where(c => c.DatabaseLevel == false).Select(c => c.Id)),

```
string.Join(",", counters.Where(c => c.DatabaseLevel).Select(c => c.Id))

);

}
```

```
public async Task<List<PerformanceAnalysisData>> GetAllAfterDateTimeStamp(int dateTimeStamp,
CancellationTokentoken cancellationToken)

{

    var transactions = new List<PerformanceAnalysisData>();

    using (var connection = _connectionFactory.Create(_consulId, _database))

    {

        var intervals = await GetIntervals(connection, cancellationToken);

        using (var command = connection.CreateCommand())

        {

            var minDateTimeStamp = 0;

            dateTimeStamp = dateTimeStamp > minDateTimeStamp ? dateTimeStamp : minDateTimeStamp;

            var keyParam = command.CreateParameter();

            keyParam.ParameterName = "@Last_Date_Time_Stamp";

            keyParam.DbType = System.Data.DbType.Int32;

            keyParam.Direction = System.Data.ParameterDirection.Input;

            keyParam.Value = dateTimeStamp;

            var countParam = command.CreateParameter();

            countParam.ParameterName = "@Count";
```

```
countParam.DbType = System.Data.DbType.Int32;

countParam.Direction = System.Data.ParameterDirection.Input;

countParam.Value = EtlSettings.BatchSize(50000);


command.CommandType = System.Data.CommandType.Text;

command.CommandText = _sql;


command.Parameters.Add(keyParam);

command.Parameters.Add(countParam);


await connection.OpenAsync();

var reader = await command.ExecuteReaderAsync();

if (reader.HasRows)
{
    while (await reader.ReadAsync(cancellationToken))
    {
        var connectionId = await Get<short>(reader, 0);

        var counterId = await Get<short>(reader, 1);

        var endSentryTimeStamp = await Get<int>(reader, 3);

        var interval = intervals[counterId];

        var server = _servers[connectionId];

        var instanceName = await Get<string>(reader, 2);

        var metricValue = Convert.ToSingle(await Get<double>(reader, 4));

        var sampleCount = await Get<int>(reader, 5);
```

```
transactions.Add(new PerformanceAnalysisData
```

```
{
```

```
    Event_Source_Connection_Id = connectionId,
```

```
    Performance_Analysis_Counter_Id = counterId,
```

```
    Instance_Name = instanceName,
```

```
    Start_Sentry_Timestamp = endSentryTimeStamp - interval,
```

```
    End_Sentry_Timestamp = endSentryTimeStamp,
```

```
    Metric_Value = metricValue,
```

```
    Sample_Count = sampleCount,
```

```
    Machine_Name = server.Machine_Name,
```

```
    Server_Instance_Name = server.Instance_Name
```

```
});
```

```
}
```

```
}
```

```
connection.Close();
```

```
}
```

```
}
```

```
return transactions;
```

```
}
```

```
private async Task<Dictionary<short, long>> GetIntervals(SqlConnection connection, CancellationToken  
cancellationToken)
```

```
{
```

```
var intervals = new Dictionary<short, long>();
```

```
using (var command = connection.CreateCommand())
```

```
{
```

```
    command.CommandText = _intervalSql;
```

```
    await connection.OpenAsync();
```

```
    var reader = await command.ExecuteReaderAsync();
```

```
    if (reader.HasRows)
```

```
    {
```

```
        while (await reader.ReadAsync(cancellationToken))
```

```
        {
```

```
            var counterId = await Get<short>(reader, 0);
```

```
            var interval = Convert.ToInt32(await Get<long>(reader, 1));
```

```
            intervals[counterId] = interval;
```

```
        }
```

```
    }
```

```
    connection.Close();
```

```
}
```

```
return intervals;
```

```
}
```

```
private async Task<T> Get<T>(SqlDataReader reader, int ordinal)
```

```

{
    var value = reader.GetValue(ordinal);

    try
    {
        return (await reader.IsDBNullAsync(ordinal)) ? default(T) : (T)reader.GetValue(ordinal);
    }

    catch (Exception ex)
    {
        return default(T);
    }
}
}
}

```

WarehouseLoader.cs (Monitoring.ETL.Domain\Fact\DatabasePerformance\WarehouseLoader.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.Fact.DatabasePerformance
```

```

{
    public class WarehouseLoader : Loader<Warehouse.Model.Performance_Analysis_Data_w>
    {
        public WarehouseLoader() : base("Fact_Database_Performance_Add")
        {
        }
    }
}
}

```


ContainersAddedModel.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ContainersAdded\ContainersAddedModel.cs):

```
using System;
```

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ContainersAdded
```

```
{  
  
    public class ContainersAddedModel : IExtractModel, SqlJson.IJsonExtractModel  
  
    {  
  
        public string JsonMessage { get; set; }  
  
        public DateTime RecordDate { get; set; }  
  
    }  
}
```

ContainersDelayFactory.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ContainersAdded\ContainersDelayFactory.cs):

```
using System;
```

```
using System.Configuration;
```

```
namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ContainersAdded
```

```
{  
  
    public class ContainersDelayFactory : ThresholdExtractionDelayFactory<ContainersAddedModel>  
  
    {  
  
        private readonly int _maxThresholdForDelay;  
  
  
        protected override TimeSpan DelayTimeSpan { get; }
```

```
protected override int MaxThresholdForDelay => _maxThresholdForDelay;
```

```
public ContainersDelayFactory()
{
    _maxThresholdForDelay = EtlSettings.ThresholdCount(1);

    DelayTimeSpan = new TimeSpan(EtlSettings.ThresholdDelayHours(),
        EtlSettings.ThresholdDelayMinutes(),
        EtlSettings.ThresholdDelaySeconds(600));
}
}
}
```

ContainersExtractor.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ContainersAdded\ContainersExtractor.cs):

```
using ETL.Process;

using Monitoring.ETL.Domain.ElasticSearch;

using Monitoring.ETL.Process;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using System.Configuration;
```

```
namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ContainersAdded
```

```
{
```

```

public class ContainersExtractor : IExtractor<ContainersAddedModel>
{
    private readonly LoadStateRepository<ContainersAddedModel> _loadStateRepository;

    private readonly ContainersRepository _repository;

    private DateTime _lastDate;

    private int maxRecordsToLoad = 100000;

    private const double NumberMinutesToLoad = 60;

    private DateTime _minDate = new DateTime(2017, 12, 31, 23, 59, 59);

    private readonly bool _isDevEnvironment;

    public ContainersExtractor()
    {
        if (DateTime.TryParse(ConfigurationManager.AppSettings["ImpactAnalysisLastLoadTime"],
            out DateTime startingDate))
        {
            _minDate = startingDate;
        }

        var serverId = EtlSettings.ServerId;

        _loadStateRepository = new LoadStateRepository<ContainersAddedModel>(serverId);

        _repository = new ContainersRepository(serverId);

        _isDevEnvironment = (EtlSettings.Environment == "t" || EtlSettings.Environment == "d");
    }

    public async Task<ResultSet<ContainersAddedModel>> ExtractAllSinceLastExtractAsync(CancellationTok
cancellationToken, IEtlProcessLogger logger)
    {

```

```
var resultSet = new ResultSet<ContainersAddedModel>();
```

```
var now = DateTime.Now;
```

```
now = now.Date.AddHours(now.Hour).AddMinutes(now.Minute).AddSeconds(now.Second); //round to nearest
```

```
minute
```

```
//If we are running in dev, then don't load data from 24 hours ago or less
```

```
if (!_isDevEnvironment)
```

```
    now = now.AddDays(-1);
```

```
var numberMinutesToLoad = NumberMinutesToLoad;
```

```
try
```

```
{
```

```
    if (_lastDate == default)
```

```
    {
```

```
        _lastDate = _loadStateRepository.GetDate();
```

```
        if (_lastDate.Year < _minDate.Year)
```

```
            _lastDate = _minDate;
```

```
    }
```

```
//we try to load numberMinutesToLoad, but we don't want it set to the future
```

```
if (numberMinutesToLoad > (now - _lastDate).TotalMinutes)
```

```
    numberMinutesToLoad = (int)(now - _lastDate).TotalMinutes;
```

```
if (_lastDate.AddMinutes(numberMinutesToLoad) > now || numberMinutesToLoad <= 0)

{

    _lastDate = _loadStateRepository.GetDate();

    return resultSet;

}
```

```
List<ContainersAddedModel> transactions = null;
```

```
while (_lastDate.AddMinutes(numberMinutesToLoad) <= now && !resultSet.Results.Any())

{
```

```
    transactions = await _repository.GetAllAfterDate(_lastDate, (int)numberMinutesToLoad,
maxRecordsToLoad, logger, cancellationToken);
```

```
    logger.Information($"Loading from: {_lastDate}, Max Minutes: {numberMinutesToLoad}, Max Records:
{maxRecordsToLoad}, Count: {transactions.Count}");
```

```
    foreach (var transaction in transactions)
```

```
    {

        resultSet.Results.Add(transaction);

    }
```

```
    if (!resultSet.Results.Any())
```

```
        _lastDate = _lastDate.AddMinutes(numberMinutesToLoad);

}
```

```
var nextDateToLoad = _lastDate.AddMinutes(numberMinutesToLoad);
```

```
var maxInResultSet = nextDateToLoad;
```

```

if (transactions != null && transactions.Any())
{
    maxInResultSet = transactions.Select(d => d.RecordDate).Max();

    logger.Information($"Loaded to: {maxInResultSet}");
}

//a little safety measure so we don't skip over anything

if (maxInResultSet < nextDateToLoad)

    nextDateToLoad = maxInResultSet;

if (nextDateToLoad > now)

    nextDateToLoad = now;

if (nextDateToLoad > _lastDate)
{
    logger.Information($"Setting Next Date To Load to {nextDateToLoad}");

    _lastDate = nextDateToLoad;

    await _loadStateRepository.SetDate(_lastDate);
}
}

catch (Exception ex)
{
    resultSet.Exceptions.Add(ex);
}

```

```

        return resultSet;
    }

    public Task<IEnumerable<ContainersAddedModel>> ExtractBetweenAsync(DateTime startDate, DateTime
endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)
    {
        throw new NotImplementedException();
    }
}
}
}

```

ContainersRepository.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ContainersAdded\ContainersRepository.cs):

```

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Threading;

using System.Threading.Tasks;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ContainersAdded
{
    /// <summary>
    /// Load the RabbitMQ With Containers added for a given N. This ETL needs to have an instance for each N
    /// </summary>

    internal class ContainersRepository

```

```

{

    private readonly ConsulConnectionFactory _connectionFactory;

    private readonly string _consulId;

    private const string Database = "Part";


    public ContainersRepository(string consulId)

    {

        _connectionFactory = new ConsulConnectionFactory();

        _consulId = consulId;

    }


    public async Task<List<ContainersAddedModel>> GetAllAfterDate(DateTime lastDate, int numberMinutesToLoad,
int maxRecordsToLoad,

        IETIProcessLogger logger, CancellationToken cancellationToken)

    {

        var transactions = new List<ContainersAddedModel>();

        using (var connection = _connectionFactory.Create(_consulId, Database))

        using (var command = connection.CreateCommand())

        {

            logger.Information($"Connecting to {_consulId} at {connection.ConnectionString}");

            command.CommandType = System.Data.CommandType.Text;


            var lastDateParam = command.CreateParameter();

            lastDateParam.ParameterName = "@Last_Date";

            lastDateParam.DbType = System.Data.DbType.DateTime;

```



```
lastDateParam.Direction = System.Data.ParameterDirection.Input;
```

```
lastDateParam.Value = lastDate;
```

```
var numMinutesToLoadParam = command.CreateParameter();
```

```
numMinutesToLoadParam.ParameterName = "@numberMinutesToLoad";
```

```
numMinutesToLoadParam.DbType = System.Data.DbType.Int32;
```

```
numMinutesToLoadParam.Direction = System.Data.ParameterDirection.Input;
```

```
numMinutesToLoadParam.Value = numberMinutesToLoad;
```

```
var countParam = command.CreateParameter();
```

```
countParam.ParameterName = "@Count";
```

```
countParam.DbType = System.Data.DbType.Int32;
```

```
countParam.Direction = System.Data.ParameterDirection.Input;
```

```
countParam.Value = maxRecordsToLoad;
```

```
command.Parameters.Add(lastDateParam);
```

```
command.Parameters.Add(countParam);
```

```
command.Parameters.Add(numMinutesToLoadParam);
```

```
command.CommandText = @"
```

```
USE Part;
```

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

```
DECLARE @End_Date DATETIME = DATEADD(MINUTE, @numberMinutesToLoad, @Last_Date);
```

```
if (@End_Date > GetDate())
```

BEGIN

set @End_Date=GetDate()

END

SELECT

@End_Date = CC.Change_Date

FROM Plexus_System.dbo.Resident_Plexus_Customer AS RPC

JOIN Part.dbo.Container_Change2 AS CC WITH (FORCESEEK(IX_History2(Plexus_Customer_No, Change_Date)))

ON CC.Plexus_Customer_No = RPC.PCN

AND CC.Change_Date > @Last_Date

AND CC.Change_Date < @End_Date

ORDER BY

CC.Change_Date

OFFSET (@Count - 1) ROWS FETCH NEXT (1) ROWS ONLY

OPTION(FORCE ORDER);

WITH Results AS

(

SELECT

C.Plexus_Customer_No AS PCN,

C.Container_Key,

C.Add_Date,

1 AS Containers_Added

FROM Plexus_System.dbo.Resident_Plexus_Customer AS RPC

JOIN Part.dbo.Container_Change2 AS CC WITH (FORCESEEK(IX_History2(Plexus_Customer_No, Change_Date)))

ON CC.Plexus_Customer_No = RPC.PCN

```

AND CC.Change_Date > @Last_Date

AND CC.Change_Date <= @End_Date

JOIN Part.dbo.Container AS C WITH (FORCESEEK(PK_Container(Plexus_Customer_No, Serial_No)))

ON C.Plexus_Customer_No = CC.Plexus_Customer_No

AND C.Serial_No = CC.Serial_No

AND C.Add_Date > @Last_Date

AND C.Add_Date <= @End_Date

)

SELECT DISTINCT TOP(@Count)

(

SELECT

C.PCN,

C.Container_Key,

C.Add_Date AS Record_Date,

C.Containers_Added,

@@SERVERNAME AS SQL_Server

FOR JSON PATH,

WITHOUT_ARRAY_WRAPPER

) AS JSON_Message,

C.Add_Date AS Record_Date

FROM Results AS C

OPTION(FORCE ORDER);";

```

```
await connection.OpenAsync(cancellationTokens);
```

```
var reader = await command.ExecuteReaderAsync(cancellationTokens);
```

```

    if (reader.HasRows)
    {
        while (await reader.ReadAsync(cancellationToken))
        {
            transactions.Add(new ContainersAddedModel
            {
                JsonMessage = await Get<string>(reader, 0),
                RecordDate = await Get<DateTime>(reader, 1)
            });
        }
    }

    connection.Close();
}

return transactions;
}

private async Task<T> Get<T>(SqlDataReader reader, int ordinal)
{
    try
    {
        return (await reader.IsDBNullAsync(ordinal)) ? default : (T)reader.GetValue(ordinal);
    }
    catch (Exception)
    {

```

```

        return default;
    }
}
}
}

```

ContainersTransformer.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ContainersAdded\ContainersTransformer.cs):

```

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

```

```

namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ContainersAdded

```

```

{

    public class ContainersTransformer : ITransformer<ContainersAddedModel, RabbitMQ.Model.ImpactAnalysis>

    {

        public async Task<IEnumerable<RabbitMQ.Model.ImpactAnalysis>>

```

```

TransformAsync(IEnumerable<ContainersAddedModel> extracted, CancellationToken cancellationToken,
    IEtlProcessLogger logger)

```

```

    {

        await Task.Yield();

        return extracted.Select(pc =>

            new RabbitMQ.Model.ImpactAnalysis

```

```
{  
  
    JsonMessage = pc.JsonMessage  
  
    });  
  
}  
  
}  
  
}
```

ProductionAddedModel.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ProductionAdded\ProductionAddedModel.cs):

```
using System;
```

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ProductionAdded
```

```
{  
  
    public class ProductionAddedModel : IExtractModel, SqlJson.IJsonExtractModel  
  
    {  
  
        public string JsonMessage { get; set; }  
  
        public DateTime RecordDate { get; set; }  
  
    }  
  
}
```

ProductionDelayFactory.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ProductionAdded\ProductionDelayFactory.cs):

```
using System;
```

```
using System.Configuration;
```

```
namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ProductionAdded
```

```
{
```

```

public class ProductionDelayFactory : ThresholdExtractionDelayFactory<ProductionAddedModel>
{
    private TimeSpan _delayTimespan;

    private int _maxThresholdForDelay;

    protected override TimeSpan DelayTimeSpan => _delayTimespan;

    protected override int MaxThresholdForDelay => _maxThresholdForDelay;

    public ProductionDelayFactory()
    {
        _maxThresholdForDelay = EtlSettings.ThresholdCount(1);

        _delayTimespan = new TimeSpan(EtlSettings.ThresholdDelayHours(),
            EtlSettings.ThresholdDelayMinutes(),
            EtlSettings.ThresholdDelaySeconds(600));
    }
}

```

ProductionExtractor.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ProductionAdded\ProductionExtractor.cs):

```

using ETL.Process;

using Monitoring.ETL.Domain.ElasticSearch;

using Monitoring.ETL.Process;

using System;

using System.Collections.Generic;

using System.Linq;

```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using System.Configuration;
```

```
namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ProductionAdded
```

```
{
```

```
    public class ProductionExtractor : IExtractor<ProductionAddedModel>
```

```
    {
```

```
        private readonly LoadStateRepository<ProductionAddedModel> _loadStateRepository;
```

```
        private readonly ProductionRepository _repository;
```

```
        private DateTime _lastDate;
```

```
        private int maxRecordsToLoad = 10000;
```

```
        private const double NumberMinutesToLoad = 60;
```

```
        private DateTime _minDate = new DateTime(2017, 12, 31, 23, 59, 59);
```

```
        private readonly bool _isDevEnvironment;
```

```
        public ProductionExtractor()
```

```
        {
```

```
            if (DateTime.TryParse(ConfigurationManager.AppSettings["ImpactAnalysisLastLoadTime"],
```

```
                out DateTime startingDate))
```

```
                _minDate = startingDate;
```

```
            _loadStateRepository = new LoadStateRepository<ProductionAddedModel>(EtlSettings.ServerId);
```

```
            _repository = new ProductionRepository(EtlSettings.ServerId);
```

```
            _isDevEnvironment = (EtlSettings.Environment == "t" || EtlSettings.Environment == "d");
```



```
}
```

```
public async Task<ResultSet<ProductionAddedModel>> ExtractAllSinceLastExtractAsync(CancellationTokencancellationToken, IEtlProcessLogger logger)

{

    var resultSet = new ResultSet<ProductionAddedModel>();

    var now = DateTime.Now;

    now = now.Date.AddHours(now.Hour).AddMinutes(now.Minute).AddSeconds(now.Second); //round to nearest
minute

    //If we are running in dev, then don't load data from 24 hours ago or less

    if (!_isDevEnvironment)

        now = now.AddDays(-1);

    var numberMinutesToLoad = NumberMinutesToLoad;

    try

    {

        if (_lastDate == default(DateTime))

        {

            _lastDate = _loadStateRepository.GetDate();

            if (_lastDate.Year < _minDate.Year)

                _lastDate = _minDate;

        }

    }

}
```

```
}
```

```
//we try to load _numberMinutesToLoad, but we don't want it set to the future
```

```
if (numberMinutesToLoad > (now - _lastDate).TotalMinutes)
```

```
    numberMinutesToLoad = (int)(now - _lastDate).TotalMinutes;
```

```
if (_lastDate.AddMinutes(numberMinutesToLoad) > now || numberMinutesToLoad <= 0)
```

```
{
```

```
    _lastDate = _loadStateRepository.GetDate();
```

```
    return resultSet;
```

```
}
```

```
List<ProductionAddedModel> transactions = null;
```

```
while (_lastDate.AddMinutes(numberMinutesToLoad) <= now && !resultSet.Results.Any())
```

```
{
```

```
    transactions = await _repository.GetAllAfterDate(_lastDate, (int)numberMinutesToLoad,
```

```
maxRecordsToLoad, cancellation_token);
```

```
    logger.Information($"Loading from: {_lastDate}, Max Minutes: {numberMinutesToLoad}, Max Records:
```

```
{maxRecordsToLoad}, Count: {transactions.Count}");
```

```
foreach (var transaction in transactions)
```

```
{
```

```
    resultSet.Results.Add(transaction);
```

```
}
```

```
if (!resultSet.Results.Any())

    _lastDate = _lastDate.AddMinutes(numberMinutesToLoad);

}

var nextDateToLoad = _lastDate.AddMinutes(numberMinutesToLoad);

var maxInResultSet = nextDateToLoad;

if (transactions != null && transactions.Any())

{

    maxInResultSet = transactions.Select(d => d.RecordDate).Max();

    logger.Information($"Loaded to: {maxInResultSet}");

}

//a little safety measure so we don't skip over anything

if (maxInResultSet < nextDateToLoad)

    nextDateToLoad = maxInResultSet;

if (nextDateToLoad > now)

    nextDateToLoad = now;

if (nextDateToLoad > _lastDate)

{

    logger.Information($"Setting Next Date To Load to {nextDateToLoad}");

    _lastDate = nextDateToLoad;

    await _loadStateRepository.SetDate(_lastDate);
```

```

        }

    }

    catch (Exception ex)

    {

        resultSet.Exceptions.Add(ex);

    }


    return resultSet;

}


public Task<IEnumerable<ProductionAddedModel>> ExtractBetweenAsync(DateTime startDate, DateTime
endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)

{

    throw new NotImplementedException();

}

}

}

```

ProductionRepository.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ProductionAdded\ProductionRepository.cs):

```

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Threading;

using System.Threading.Tasks;

```

```

namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ProductionAdded

```

```

{

/// <summary>

/// Load the RabbitMQ With Production for a given N. This ETL needs to have an instance for each N

/// </summary>


internal class ProductionRepository

{

    private readonly ConsulConnectionFactory _connectionFactory;

    private readonly string _consulId;

    private const string Database = "Part";


    public ProductionRepository(string consulId)

    {

        _connectionFactory = new ConsulConnectionFactory();

        _consulId = consulId;

    }


    public async Task<List<ProductionAddedModel>> GetAllAfterDate(DateTime lastDate, int numberMinutesToLoad,

int maxRecordsToLoad,

    CancellationToken cancellationToken)

    {

        var transactions = new List<ProductionAddedModel>();


        using (var connection = _connectionFactory.Create(_consulId, Database))

        using (var command = connection.CreateCommand())

        {

```

```
command.CommandType = System.Data.CommandType.Text;
```

```
var lastDateParam = command.CreateParameter();
```

```
lastDateParam.ParameterName = "@Last_Date";
```

```
lastDateParam.DbType = System.Data.DbType.DateTime;
```

```
lastDateParam.Direction = System.Data.ParameterDirection.Input;
```

```
lastDateParam.Value = lastDate;
```

```
var numMinutesToLoadParam = command.CreateParameter();
```

```
numMinutesToLoadParam.ParameterName = "@numberMinutesToLoad";
```

```
numMinutesToLoadParam.DbType = System.Data.DbType.Int32;
```

```
numMinutesToLoadParam.Direction = System.Data.ParameterDirection.Input;
```

```
numMinutesToLoadParam.Value = numberMinutesToLoad;
```

```
var countParam = command.CreateParameter();
```

```
countParam.ParameterName = "@Count";
```

```
countParam.DbType = System.Data.DbType.Int32;
```

```
countParam.Direction = System.Data.ParameterDirection.Input;
```

```
countParam.Value = maxRecordsToLoad;
```

```
command.Parameters.Add(lastDateParam);
```

```
command.Parameters.Add(countParam);
```

```
command.Parameters.Add(numMinutesToLoadParam);
```

```
command.CommandText = @"
```

USE Part;

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

DECLARE @End_Date DATETIME = DATEADD(MINUTE, @numberMinutesToLoad, @Last_Date);

if (@End_Date > GetDate())

BEGIN

set @End_Date=GetDate()

END

SELECT

@End_Date = P.Record_Date

FROM

(

SELECT

P.Record_Date

FROM Plexus_System.dbo.Resident_Plexus_Customer AS RPC

JOIN Part.dbo.Production AS P WITH (FORCESEEK(IX_Workcenter_Key_Record_Date(Plexus_Customer_No,
Workcenter_Key, Record_Date)))

ON P.Plexus_Customer_No = RPC.PCN

AND P.Workcenter_Key IS NULL

AND P.Record_Date > @Last_Date

AND P.Record_Date < @End_Date

UNION ALL

```

SELECT

    P.Record_Date

FROM Plexus_System.dbo.Resident_Plexus_Customer AS RPC

JOIN Part.dbo.Workcenter AS W WITH (FORCESEEK(PK_Workcenter(Plexus_Customer_No)))

    ON W.Plexus_Customer_No = RPC.PCN

JOIN Part.dbo.Production AS P WITH (FORCESEEK(IX_Workcenter_Key_Record_Date(Plexus_Customer_No,
Workcenter_Key, Record_Date)))

    ON P.Plexus_Customer_No = W.Plexus_Customer_No

    AND P.Workcenter_Key = W.Workcenter_Key

    AND P.Record_Date > @Last_Date

    AND P.Record_Date < @End_Date

) AS P

ORDER BY

    P.Record_Date

OFFSET (@Count - 1) ROWS FETCH NEXT (1) ROWS ONLY

OPTION(FORCE ORDER);

```

WITH Results AS

```

(

SELECT

    P.Plexus_Customer_No AS PCN,

    P.Production_No,

    P.Record_Date,

    P.Quantity AS Production_Added

FROM Plexus_System.dbo.Resident_Plexus_Customer AS RPC

JOIN Part.dbo.Workcenter AS W WITH (FORCESEEK(PK_Workcenter(Plexus_Customer_No)))

```



```

ON W.Plexus_Customer_No = RPC.PCN

JOIN Part.dbo.Production AS P WITH (FORCESEEK(IX_Workcenter_Key_Record_Date(Plexus_Customer_No,
Workcenter_Key, Record_Date)))

ON P.Plexus_Customer_No = W.Plexus_Customer_No

AND P.Workcenter_Key = W.Workcenter_Key

AND P.Record_Date > @Last_Date

AND P.Record_Date <= @End_Date

)

SELECT DISTINCT TOP(@Count)

(

SELECT

C.PCN,

C.Production_No,

C.Record_Date,

C.Production_Added,

@@SERVERNAME AS SQL_Server

FOR JSON PATH,

WITHOUT_ARRAY_WRAPPER

) AS JSON_Message,

C.Record_Date AS Record_Date

FROM Results AS C

OPTION(FORCE ORDER);";

```

```

await connection.OpenAsync(cancellationToken);

```

```

var reader = await command.ExecuteReaderAsync(cancellationToken);

```

```

    if (reader.HasRows)
    {
        while (await reader.ReadAsync(cancellationToken))
        {
            transactions.Add(new ProductionAddedModel
            {
                JsonMessage = await Get<string>(reader, 0),
                RecordDate = await Get<DateTime>(reader, 1)
            });
        }
    }

    connection.Close();
}

return transactions;
}

private async Task<T> Get<T>(SqlDataReader reader, int ordinal)
{
    try
    {
        return (await reader.IsDBNullAsync(ordinal)) ? default(T) : (T)reader.GetValue(ordinal);
    }
    catch (Exception ex)
    {

```

```

        return default(T);
    }
}
}
}

```

ProductionTransformer.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ProductionAdded\ProductionTransformer.cs):

```

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

```

```

namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ProductionAdded

```

```

{

    public class ProductionTransformer : ITransformer<ProductionAddedModel, RabbitMQ.Model.ImpactAnalysis>

    {

        public async Task<IEnumerable<RabbitMQ.Model.ImpactAnalysis>>

TransformAsync(IEnumerable<ProductionAddedModel> extracted, CancellationToken cancellationToken,

IEtlProcessLogger logger)

        {

            await Task.Yield();

            return extracted.Select(pc =>

                new RabbitMQ.Model.ImpactAnalysis

```

```
{  
  
    JsonMessage = pc.JsonMessage  
  
    });  
  
}  
  
}  
  
}
```

DelayFactory.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\RMQExtractor\DelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.RMQExtractor
```

```
{  
  
    public class DelayFactory : ThresholdExtractionDelayFactory<RabbitMQ.Model.ImpactAnalysis>  
  
    {  
  
        protected override TimeSpan DelayTimeSpan => new TimeSpan(0, 0, 5);  
  
        protected override int MaxThresholdForDelay => 1000;  
  
    }  
  
}
```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\RMQExtractor\RabbitMQExtractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.RMQExtractor
```

```
{  
  
    public sealed class RabbitMQExtractor : IExtractor<RabbitMQ.Model.ImpactAnalysis>
```

```
{  
  
    private RabbitMQ.ImpactAnalysis.Consumer _consumer;
```

```
  
    public async Task<ResultSet<RabbitMQ.Model.ImpactAnalysis>> ExtractAllSinceLastExtractAsync(CancellationToken  
cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    _consumer = _consumer ?? new RabbitMQ.ImpactAnalysis.Consumer(logger);  
  
    return await _consumer.ExtractAsync();  
  
}
```

```
  
    public Task<IEnumerable<RabbitMQ.Model.ImpactAnalysis>> ExtractBetweenAsync(DateTime startDate, DateTime  
endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    throw new NotImplementedException();  
  
}  
  
}  
  
}
```

RabbitMQLoader.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\RMQExtractor\RabbitMQLoader.cs):

```
namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.RMQExtractor
```

```

{

public class RabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.ImpactAnalysis>

{

public RabbitMQLoader() : base(new RabbitMQ.ImpactAnalysis.Producer())

{

}

}

}

```

Transformer.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\RMQExtractor\Transformer.cs):

```

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;


using ETL.Process;


using Monitoring.ETL.Process;


namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.RMQExtractor

{

/// <summary>

/// Load the messages from RabbitMQ into the table Impact_Analysis_w.

/// </summary>

public class Transformer : ITransformer<RabbitMQ.Model.ImpactAnalysis, Warehouse.Model.Impact_Analysis_w>

{

```

```

public async Task<IEnumerable<Warehouse.Model.Impact_Analysis_w>>
TransformAsync(IEnumerable<RabbitMQ.Model.ImpactAnalysis> extracted, CancellationToken cancellationToken,
IEtlProcessLogger logger)
{
    await Task.Yield();

    return extracted.Select(pc => new Warehouse.Model.Impact_Analysis_w
    {
        JSON_Message = pc.JsonMessage
    });
}
}
}

```

WarehouseLoader.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\RMQExtractor\WarehouseLoader.cs):

```

using Monitoring.ETL.Domain.Warehouse;

using Monitoring.ETL.Domain.Warehouse.Model;

namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.RMQExtractor
{
    /// <summary>
    /// Execute the Fact_Impact_Analysis_add which takes the JSON from Impact_Analysis_w and loads the
    Fact_Impact_Analysis table.
    /// This sproc loads all messages, for Shippers, Containers, and Production
    /// </summary>

    public class WarehouseLoader : Loader<Impact_Analysis_w>

```

```

{

    public WarehouseLoader() : base("Fact_Impact_Analysis_Add", true)

    {

    }

}

}

```

ShippersDelayFactory.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ShippersShipped\ShippersDelayFactory.cs):

```
using System;
```

```
using System.Configuration;
```

```
namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ShippersShipped
```

```

{

    public class ShippersDelayFactory : ThresholdExtractionDelayFactory<ShippersShippedModel>

    {

        private readonly int _maxThresholdForDelay;

        protected override TimeSpan DelayTimeSpan { get; }

        protected override int MaxThresholdForDelay => _maxThresholdForDelay;

        public ShippersDelayFactory()

        {

            _maxThresholdForDelay = EtlSettings.ThresholdCount(1);

            DelayTimeSpan = new TimeSpan(EtlSettings.ThresholdDelayHours(),

                EtlSettings.ThresholdDelayMinutes(),

```



```
        EtlSettings.ThresholdDelaySeconds(600));  
  
    }  
  
}  
  
}
```

ShippersExtractor.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ShippersShipped\ShippersExtractor.cs):

```
using ETL.Process;  
  
using Monitoring.ETL.Domain.ElasticSearch;  
  
using Monitoring.ETL.Process;  
  
using System;  
  
using System.Collections.Generic;  
  
using System.Linq;  
  
using System.Threading;  
  
using System.Threading.Tasks;  
  
using System.Configuration;  
  
  
namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ShippersShipped  
{  
  
    public class ShippersExtractor : IExtractor<ShippersShippedModel>  
    {  
  
        private readonly LoadStateRepository<ShippersShippedModel> _loadStateRepository;  
  
        private readonly ShippersRepository _repository;  
  
        private DateTime _lastDate;  
  
        private int maxRecordsToLoad = 10000;  
  
        private const double NumberMinutesToLoad = 60;  
  
        private DateTime _minDate = new DateTime(2017, 12, 31, 23, 59, 59);  
  
    }  
}
```

```
private readonly bool _isDevEnvironment;
```

```
public ShippersExtractor()
```

```
{
```

```
    if (DateTime.TryParse(ConfigurationManager.AppSettings["ImpactAnalysisLastLoadTime"],
```

```
        out DateTime startingDate))
```

```
        _minDate = startingDate;
```

```
    _loadStateRepository = new LoadStateRepository<ShippersShippedModel>(EtlSettings.ServerId);
```

```
    _repository = new ShippersRepository(EtlSettings.ServerId);
```

```
    _isDevEnvironment = (EtlSettings.Environment == "t" || EtlSettings.Environment == "d");
```

```
}
```

```
public async Task<ResultSet<ShippersShippedModel>> ExtractAllSinceLastExtractAsync(CancellationTokencancellationTokentoken, IEtlProcessLogger logger)
```

```
{
```

```
    var resultSet = new ResultSet<ShippersShippedModel>();
```

```
    var now = DateTime.Now;
```

```
    now = now.Date.AddHours(now.Hour).AddMinutes(now.Minute).AddSeconds(now.Second); //round to nearestminute
```

```
    //If we are running in dev, then don't load data from 24 hours ago or less
```

```
    if (_isDevEnvironment)
```

```
        now = now.AddDays(-1);
```

```

var numberMinutesToLoad = NumberMinutesToLoad;

try

{

    if (_lastDate == default(DateTime))

    {

        _lastDate = _loadStateRepository.GetDate();

        if (_lastDate.Year < _minDate.Year)

            _lastDate = _minDate;

    }

    //we try to load numberMinutesToLoad, but we don't want it set to the future

    if (numberMinutesToLoad > (now - _lastDate).TotalMinutes)

        numberMinutesToLoad = (int)(now - _lastDate).TotalMinutes;

    if (_lastDate.AddMinutes(numberMinutesToLoad) > now || numberMinutesToLoad <= 0)

    {

        _lastDate = _loadStateRepository.GetDate();

        return resultSet;

    }

    List<ShippersShippedModel> transactions = null;

    while (_lastDate.AddMinutes(numberMinutesToLoad) <= now && !resultSet.Results.Any())

    {

        transactions = await _repository.GetAllAfterDate(_lastDate, (int)numberMinutesToLoad,

```

```
maxRecordsToLoad, cancellationToken);
```

```
    logger.Information($"Loading from: {_lastDate}, Max Minutes: {numberMinutesToLoad}, Max Records:  
{maxRecordsToLoad}, Count: {transactions.Count}");
```

```
    foreach (var transaction in transactions)

    {

        resultSet.Results.Add(transaction);

    }

    if (!resultSet.Results.Any())

        _lastDate = _lastDate.AddMinutes(numberMinutesToLoad);

}

var nextDateToLoad = _lastDate.AddMinutes(numberMinutesToLoad);

var maxInResultSet = nextDateToLoad;

if (transactions != null && transactions.Any())

{

    maxInResultSet = transactions.Select(d => d.RecordDate).Max();

    logger.Information($"Loaded to: {maxInResultSet}");

}

//a little safety measure so we don't skip over anything

if (maxInResultSet < nextDateToLoad)

    nextDateToLoad = maxInResultSet;
```

```
if (nextDateToLoad > now)
```

```
    nextDateToLoad = now;
```

```
if (nextDateToLoad >= _lastDate)
```

```
{
```

```
    logger.Information($"Setting Next Date To Load to {nextDateToLoad}");
```

```
    _lastDate = nextDateToLoad;
```

```
    await _loadStateRepository.SetDate(_lastDate);
```

```
}
```

```
}
```

```
catch (Exception ex)
```

```
{
```

```
    resultSet.Exceptions.Add(ex);
```

```
}
```

```
return resultSet;
```

```
}
```

```
public Task<IEnumerable<ShippersShippedModel>> ExtractBetweenAsync(DateTime startDate, DateTime  
endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
{
```

```
    throw new NotImplementedException();
```

```
}
```

```
}
```

```
}
```

ShippersRepository.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ShippersShipped\ShippersRepository.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Data.SqlClient;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ShippersShipped
```

```
{
```

```
    /// <summary>
```

```
    /// Load the RabbitMQ With Shippers shipped for a given N. This ETL needs to have an instance for each N
```

```
    /// </summary>
```

```
    internal class ShippersRepository
```

```
    {
```

```
        private readonly ConsulConnectionFactory _connectionFactory;
```

```
        private readonly string _consulId;
```

```
        private const string Database = "Part";
```

```
        public ShippersRepository(string consulId)
```

```
        {
```

```
            _connectionFactory = new ConsulConnectionFactory();
```

```
            _consulId = consulId;
```

```
        }
```

```
        public async Task<List<ShippersShippedModel>> GetAllAfterDate(DateTime lastDate, int numberMinutesToLoad,
```

```

int maxRecordsToLoad, CancellationToken cancellationToken)

{

    var transactions = new List<ShippersShippedModel>();


    using (var connection = _connectionFactory.Create(_consulId, Database))

    using (var command = connection.CreateCommand())

    {

        command.CommandType = System.Data.CommandType.Text;


        var lastDateParam = command.CreateParameter();

        lastDateParam.ParameterName = "@Last_Date";

        lastDateParam.DbType = System.Data.DbType.DateTime;

        lastDateParam.Direction = System.Data.ParameterDirection.Input;

        lastDateParam.Value = lastDate;


        var numMinutesToLoadParam = command.CreateParameter();

        numMinutesToLoadParam.ParameterName = "@numberMinutesToLoad";

        numMinutesToLoadParam.DbType = System.Data.DbType.Int32;

        numMinutesToLoadParam.Direction = System.Data.ParameterDirection.Input;

        numMinutesToLoadParam.Value = numberMinutesToLoad;


        var countParam = command.CreateParameter();

        countParam.ParameterName = "@Count";

        countParam.DbType = System.Data.DbType.Int32;

        countParam.Direction = System.Data.ParameterDirection.Input;

        countParam.Value = maxRecordsToLoad;
    }
}

```

```
command.Parameters.Add(lastDateParam);
```

```
command.Parameters.Add(countParam);
```

```
command.Parameters.Add(numMinutesToLoadParam);
```

```
command.CommandText = @"
```

```
USE Part;
```

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

```
DECLARE @End_Date DATETIME = DATEADD(minute, @numberMinutesToLoad, @Last_Date);
```

```
if (@End_Date > GetDate())
```

```
BEGIN
```

```
set @End_Date=GetDate()
```

```
END
```

```
SELECT
```

```
@End_Date = S.Ship_Date
```

```
FROM Plexus_System.dbo.Resident_Plexus_Customer AS RPC
```

```
JOIN Sales.dbo.Shipper AS S WITH (FORCESEEK(IX_Ship_Date_Customer_No(PCN, Ship_Date)))
```

```
ON S.PCN = RPC.PCN
```

```
AND S.Ship_Date > @Last_Date
```

```
AND S.Ship_Date < @End_Date
```

```
AND S.Ship_Date < DATEADD(MINUTE, @numberMinutesToLoad, GETDATE())
```

```
ORDER BY
```


S.Ship_Date

OFFSET (@Count - 1) ROWS FETCH NEXT (1) ROWS ONLY

OPTION(FORCE ORDER);

WITH Results AS

(

SELECT

S.PCN

,S.Shipper_Key

,S.Ship_Date

,1 AS Shippers_Shipped

FROM Plexus_System.dbo.Resident_Plexus_Customer AS RPC

JOIN Sales.dbo.Shipper AS S WITH (FORCESEEK(IX_Ship_Date_Customer_No(PCN, Ship_Date)))

ON S.PCN = RPC.PCN

AND S.Ship_Date > @Last_Date

AND S.Ship_Date <= @End_Date

JOIN Sales.dbo.Shipper_Status AS SS

ON SS.PCN = S.PCN

AND SS.Shipper_Status_Key = S.Shipper_Status_Key

WHERE SS.Shipped = 1

)

SELECT DISTINCT TOP(@Count)

(

SELECT

C.PCN,

C.Shipper_Key,

```

C.Ship_Date AS Record_Date,

C.Shippers_Shipped,

@@SERVERNAME AS SQL_Server

FOR JSON PATH,

WITHOUT_ARRAY_WRAPPER

) AS JSON_Message,

C.Ship_Date AS Record_Date

FROM Results AS C

OPTION(FORCE ORDER);";

```

```

await connection.OpenAsync(cancellationToken);

var reader = await command.ExecuteReaderAsync(cancellationToken);

if (reader.HasRows)
{
    while (await reader.ReadAsync(cancellationToken))
    {
        transactions.Add(new ShippersShippedModel
        {
            JsonMessage = await Get<string>(reader, 0),
            RecordDate = await Get<DateTime>(reader, 1)
        });
    }
}

connection.Close();

```

```

    }

    return transactions;
}

private async Task<T> Get<T>(SqlDataReader reader, int ordinal)
{
    try
    {
        return (await reader.IsDBNullAsync(ordinal)) ? default(T) : (T)reader.GetValue(ordinal);
    }

    catch (Exception ex)
    {
        return default(T);
    }
}
}
}
}

```

ShippersShippedModel.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ShippersShipped\ShippersShippedModel.cs):

```

using System;

using ETL.Process;

namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ShippersShipped
{
    public class ShippersShippedModel : IExtractModel, SqlJson.IJsonExtractModel

```

```

{

    public string JsonMessage { get; set; }

    public DateTime RecordDate { get; set; }

}

}

```

ShippersTransformer.cs (Monitoring.ETL.Domain\Fact\ImpactAnalysis\ShippersShipped\ShippersTransformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Fact.ImpactAnalysis.ShippersShipped
```

```

{

    public class ShippersTransformer : ITransformer<ShippersShippedModel, RabbitMQ.Model.ImpactAnalysis>

```

```

    {

        public async Task<IEnumerable<RabbitMQ.Model.ImpactAnalysis>>

```

```

TransformAsync(IEnumerable<ShippersShippedModel> extracted, CancellationToken cancellationToken,
    IEtlProcessLogger logger)

```

```

    {

        await Task.Yield();

        return extracted.Select(pc =>

            new RabbitMQ.Model.ImpactAnalysis

```

```

{
    JsonMessage = pc.JsonMessage

});

}

}

}

```

ProcedureExecution.cs (Monitoring.ETL.Domain\Fact\SqlExecution\Parsing\ProcedureExecution.cs):

```
using System.Collections.Generic;
```

```
namespace Monitoring.ETL.Domain.Fact.SqlExecution.Parsing
```

```

{

    public class ProcedureExecution

    {

        public string Database { get; set; }

        public string ProcedureName { get; set; }

        public IEnumerable<ProcedureExecutionParameter> Parameters { get; set; }

    }

}

```

ProcedureExecutionParameter.cs

(Monitoring.ETL.Domain\Fact\SqlExecution\Parsing\ProcedureExecutionParameter.cs):

```
namespace Monitoring.ETL.Domain.Fact.SqlExecution.Parsing
```

```

{

    public class ProcedureExecutionParameter

    {

```

```
        public string Name { get; set; }

        public string Value { get; set; }

    }

}
```

ProcedureExecutionParser.cs (Monitoring.ETL.Domain\Fact\SqlExecution\Parsing\ProcedureExecutionParser.cs):

```
using Microsoft.SqlServer.TransactSql.ScriptDom;
```

```
using Monitoring.ETL.Process;
```

```
using System;
```

```
using System.IO;
```

```
using System.Linq;
```

```
namespace Monitoring.ETL.Domain.Fact.SqlExecution.Parsing
```

```
{
```

```
    public class ProcedureExecutionParser
```

```
    {
```

```
        private readonly TSqlParser _parser;
```

```
        private readonly StoredProcedureRepository _procedures;
```

```
        public ProcedureExecutionParser(StoredProcedureRepository storedProcedureRepository)
```

```
        {
```

```
            _parser = new TSql150Parser(false);
```

```
            _procedures = storedProcedureRepository;
```

```
        }
```

```

public ProcedureExecution Parse(string database, string sql, IEtlProcessLogger logger)
{
    var execution = new ProcedureExecution();

    var fragment = _parser.Parse(new StringReader(sql), out var errors) as TSqIScript;

    if (!errors.Any())
    {
        var statement = fragment?.Batches.SelectMany(b =>
b.Statements).OfType<ExecuteStatement>().FirstOrDefault();

        if (statement != null)
        {
            var procedureReference = statement.ExecuteSpecification.ExecutableEntity as
ExecutableProcedureReference;

            var identifiers = procedureReference.ProcedureReference.ProcedureReference.Name.Identifiers;

            execution.ProcedureName = identifiers.Last().Value;

            execution.Database = identifiers.Count < 3 ? database : identifiers.Reverse().Skip(2).First().Value;

            execution.Parameters = procedureReference.Parameters
                .Select(p =>
                    new ProcedureExecutionParameter
                    {
                        Name = p.Variable?.Name,

                        Value = p.ParameterValue.GetValue()
                    }
                );
        }
    }
}

```

```

    }

)

.ToList();

if (execution.Parameters.Any(p => string.IsNullOrEmpty(p.Name)))
{
    try
    {
        var procedure = _procedures.Get(database, execution.ProcedureName);

        foreach (var pair in procedure.Parameters.Zip(execution.Parameters, (param, value) => new { param,
value }))
        {
            if (pair.value == null) break;

            pair.value.Name = pair.param.Name;
        }
    }
    catch (Exception ex)
    {
        logger.Information($"Error while parsing procedure parameters ({database},
{execution.ProcedureName}): {ex.Message}");
    }
}
}
}

```



```

        return execution;
    }

}

}

```

StoredProcedureRepository.cs (Monitoring.ETL.Domain\Fact\SqlExecution\Parsing\StoredProcedureRepository.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using Microsoft.SqlServer.Management.Common;
```

```
using Microsoft.SqlServer.Management.Smo;
```

```
namespace Monitoring.ETL.Domain.Fact.SqlExecution.Parsing
```

```
{
```

```
    public class StoredProcedureRepository
```

```
    {
```

```
        private readonly Server _server;
```

```
        private readonly Dictionary<string, Dictionary<string, ProcedureExecution>> _procedures;
```

```
        public StoredProcedureRepository(string serverId)
```

```
        {
```

```
            var connection = new ConsulConnectionFactory().Create(serverId, null);
```

```
            _server = new Server(new ServerConnection(connection));
```

```

    _procedures = new Dictionary<string, Dictionary<string, ProcedureExecution>>();
}

```

```

public ProcedureExecution Get(string database, string procedureName)

```

```

{
    if (_procedures.ContainsKey(database) == false)
    {
        _procedures[database] = new Dictionary<string, ProcedureExecution>();
    }

```

```

    if (_procedures[database].ContainsKey(procedureName) == false)

```

```

    {
        if (procedureName.ToLowerInvariant() == "sp_executesql")

```

```

        {
            _procedures[database][procedureName] =
                new ProcedureExecution
                {
                    ProcedureName = procedureName,
                    Parameters =
                        new List<ProcedureExecutionParameter>
                        {
                            new ProcedureExecutionParameter { Name = "@stmt" },
                            new ProcedureExecutionParameter { Name = "@params" }
                        }
                }
            .Concat(
                Enumerable

```

```

        .Range(1, 100)

        .Select(i => new ProcedureExecutionParameter { Name = "@param" + i })

    )

};

}

else

{

    var procedure =

_server.Databases[database].StoredProcedures.OfType<StoredProcedure>().SingleOrDefault(sp => sp.Name ==
procedureName);

```

```

    _procedures[database][procedureName] =

        new ProcedureExecution

        {

            ProcedureName = procedureName,

            Parameters = procedure?.Parameters?.OfType<StoredProcedureParameter>()

                .Select(p => new ProcedureExecutionParameter { Name = p.Name }) ??

                new List<ProcedureExecutionParameter>()

        };

    }

}

return _procedures[database][procedureName];

}

}

}

```

TSqlParserExtensions.cs (Monitoring.ETL.Domain\Fact\SqlExecution\Parsing\TSqlParserExtensions.cs):

```
using Microsoft.SqlServer.TransactSql.ScriptDom;
```

```
namespace Monitoring.ETL.Domain.Fact.SqlExecution.Parsing
```

```
{
```

```
    public static class TSqlParserExtensions
```

```
    {
```

```
        public static string GetValue(this TSqlFragment fragment)
```

```
        {
```

```
            switch (fragment)
```

```
            {
```

```
                case Literal literal: return literal.Value;
```

```
                default: return string.Empty;
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

RabbitMQExtractionDelayFactory.cs (Monitoring.ETL.Domain\Fact\SqlExecution\RabbitMQExtractionDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Fact.SqlExecution
```

```
{
```

```
    public class RabbitMQExtractionDelayFactory : ThresholdExtractionDelayFactory<RabbitMQ.Model.SqlExecution>
```

```
    {
```

```
        protected override int MaxThresholdForDelay => 1000;
```

```

        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);
    }
}

RabbitMQExtractor.cs (Monitoring.ETL.Domain\Fact\SqlExecution\RabbitMQExtractor.cs):

using System;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;


using ETL.Process;

using Monitoring.ETL.Process;


namespace Monitoring.ETL.Domain.Fact.SqlExecution
{
    public sealed class RabbitMQExtractor : IExtractor<RabbitMQ.Model.SqlExecution>
    {
        private RabbitMQ.SqlExecution.Consumer _consumer;


        public async Task<ResultSet<RabbitMQ.Model.SqlExecution>>

ExtractAllSinceLastExtractAsync(Cancellation token cancellationToken, IEtlProcessLogger logger)
        {
            _consumer = _consumer ?? new RabbitMQ.SqlExecution.Consumer(logger);

```

```
var result = await _consumer.ExtractAsync();

return result;

}
```

```
public Task<IEnumerable<RabbitMQ.Model.SqlExecution>> ExtractBetweenAsync(DateTime startDate, DateTime
endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)

{

    throw new NotImplementedException();

}

}

}
```

RabbitMQLoader.cs (Monitoring.ETL.Domain\Fact\SqlExecution\RabbitMQLoader.cs):

```
namespace Monitoring.ETL.Domain.Fact.SqlExecution

{

    public class RabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.SqlExecution>

    {

        public RabbitMQLoader()

            : base(new RabbitMQ.SqlExecution.Producer())

        {

        }

    }

}
```

RabbitMQWarehouseTransformer.cs (Monitoring.ETL.Domain\Fact\SqlExecution\RabbitMQWarehouseTransformer.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Configuration;
```

```
using System.IO;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.Fact.SqlExecution.Parsing;
```

```
using Monitoring.ETL.Process;
```

```
using Newtonsoft.Json.Linq;
```

```
namespace Monitoring.ETL.Domain.Fact.SqlExecution
```

```
{
```

```
    public class RabbitMQWarehouseTransformer : ITransformer<RabbitMQ.Model.SqlExecution,  
Warehouse.Model.SQL_Execution_w>
```

```
{
```

```
    private readonly ProcedureExecutionParser _parser;
```

```
    public RabbitMQWarehouseTransformer()
```

```
{
```

```
        _parser = new ProcedureExecutionParser(new StoredProcedureRepository(EtlSettings.ServerId));
```

```
}
```

```
public async Task<IEnumerable<Warehouse.Model.SQL_Execution_w>>
```

```
TransformAsync(IEnumerable<RabbitMQ.Model.SqlExecution> extracted, CancellationToken cancellationToken,
```

```
IEtlProcessLogger logger)
```

```
{  
  
    await Task.Yield();  
  
    return extracted.Select(ws =>  
  
    {  
  
        var message = JObject.Parse(ws.JsonMessage);  
  
        var database = message["Database_Name"]?.Value<string>();  
  
        var sql = message["textdata"]?.Value<string>();  
  
  
  
        var execution = _parser.Parse(database, sql, logger);  
  
  
        var pcnParam =  
  
        execution?.Parameters?.SingleOrDefault(p => p.Name == "@Plexus_Customer_No") ??  
  
        execution?.Parameters?.SingleOrDefault(p => p.Name == "@PCN") ??  
  
        execution?.Parameters?.FirstOrDefault(p => p.Name != null && p.Name.Contains("Plexus_Customer_No"))  
  
        ??  
  
        execution?.Parameters?.FirstOrDefault(p => p.Name != null && p.Name.Contains("PCN"));  
  
  
        var punParam =  
  
        execution?.Parameters?.SingleOrDefault(p => p.Name == "@Plexus_User_No") ??  
  
        execution?.Parameters?.SingleOrDefault(p => p.Name == "@PUN") ??  
  
        execution?.Parameters?.FirstOrDefault(p => p.Name != null && p.Name.Contains("Plexus_User_No")) ??  
  
        execution?.Parameters?.FirstOrDefault(p => p.Name != null && p.Name.Contains("PUN"));
```



```

        if (execution?.ProcedureName != null) message["Procedure_Name"] = execution.ProcedureName;

        if (pcnParam?.Value != null) message["PCN"] = pcnParam.Value;

        if (punParam?.Value != null) message["PUN"] = punParam.Value;

        var jsonMessage = new Warehouse.Model.SQL_Execution_w
        {
            JSON_Message = message.ToString()
        };

        return jsonMessage;
    }).ToList();
}
}
}

```

DelayFactory.cs (Monitoring.ETL.Domain\Fact\SqlExecution\Trace\DelayFactory.cs):

```

using System;

namespace Monitoring.ETL.Domain.Fact.SqlExecution.Trace
{
    public class DelayFactory : ThresholdExtractionDelayFactory<TraceDataTable>
    {
        protected override int MaxThresholdForDelay => 100;

        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);
    }
}

```

```
}
```

```
}
```

Extractor.cs (Monitoring.ETL.Domain\Fact\SqlExecution\Trace\Extractor.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.ElasticSearch;
```

```
using Monitoring.ETL.Process;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using System.Configuration;
```

```
namespace Monitoring.ETL.Domain.Fact.SqlExecution.Trace
```

```
{
```

```
    public class Extractor : IExtractor<TraceDataTable>
```

```
    {
```

```
        private readonly LoadStateRepository<TraceDataTable> _loadStateRepository;
```

```
        private readonly Repository _repository;
```

```
        private DateTime _lastDate;
```

```
        public Extractor()
```

```
        {
```

```
            var table = ConfigurationManager.AppSettings["tableName"];
```

```
            _loadStateRepository = new LoadStateRepository<TraceDataTable>(EtlSettings.ServerId);
```

```
_repository = new Repository("sqlperf", "Performance", table);  
}
```

```
public async Task<ResultSet<TraceDataTable>> ExtractAllSinceLastExtractAsync(CancellationTok  
cancellationToken, IEtlProcessLogger logger)
```

```
{  
    var resultSet = new ResultSet<TraceDataTable>();
```

```
    try
```

```
{  
    if (_lastDate == default(DateTime))  
    {  
        _lastDate = _loadStateRepository.GetDate();  
    }
```

```
    var transactions = await _repository.GetAllAfterDate(_lastDate, cancellationToken);
```

```
    foreach (var transaction in transactions)
```

```
{  
        resultSet.Results.Add(transaction);  
    }
```

```
    var max = transactions
```

```
        .Select(d => d.EndTime)
```

```
        .OrderByDescending(date => date)
```

```
        .FirstOrDefault();
```

```

        if (max > _lastDate)
        {
            _lastDate = max;

            await _loadStateRepository.SetDate(_lastDate);
        }
    }

    catch (Exception ex)
    {
        resultSet.Exceptions.Add(ex);
    }

    return resultSet;
}

```

```

public Task<IEnumerable<TraceDataTable>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
CancellationTokentoken cancellationToken, IEtlProcessLogger logger)
{
    throw new NotImplementedException();
}
}
}
}

```

RabbitMQTransformer.cs (Monitoring.ETL.Domain\Fact\SqlExecution\Trace\RabbitMQTransformer.cs):

```

using System.Collections.Generic;

using System.Linq;

using System.Threading;

```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Fact.SqlExecution.Trace
```

```
{
```

```
    public class RabbitMQTransformer : ITransformer<TraceDataTable, RabbitMQ.Model.SqlExecution>
```

```
    {
```

```
        public async Task<IEnumerable<RabbitMQ.Model.SqlExecution>>
```

```
TransformAsync(IEnumerable<TraceDataTable> extracted, CancellationToken cancellationToken, IEtlProcessLogger  
logger)
```

```
    {
```

```
        await Task.Yield();
```

```
        return extracted.Select(im => new RabbitMQ.Model.SqlExecution
```

```
        {
```

```
            JsonMessage = im.JsonMessage
```

```
        });
```

```
    }
```

```
}
```

```
}
```

Repository.cs (Monitoring.ETL.Domain\Fact\SqlExecution\Trace\Repository.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Data.SqlClient;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.Fact.SqlExecution.Trace
```

```
{
```

```
    internal class Repository
```

```
    {
```

```
        private readonly ConsulConnectionFactory _connectionFactory;
```

```
        private readonly string _consulId;
```

```
        private readonly string _database;
```

```
        private readonly string _tableName;
```

```
        public Repository(string consulId, string database, string tableName)
```

```
        {
```

```
            _connectionFactory = new ConsulConnectionFactory();
```

```
            _consulId = consulId;
```

```
            _database = database;
```

```
            _tableName = tableName;
```

```
        }
```

```
        public async Task<List<TraceDataTable>> GetAllAfterDate(DateTime date, CancellationToken cancellationToken)
```

```
        {
```

```

var transactions = new List<TraceDataTable>();

using (var connection = _connectionFactory.Create(_consulId, _database))

using (var command = connection.CreateCommand())

{

    var minDate = new DateTime(2000, 1, 1);

    date = date > minDate ? date : minDate;

    var keyParam = command.CreateParameter();

    keyParam.ParameterName = "@Last_Date";

    keyParam.DbType = System.Data.DbType.DateTime;

    keyParam.Direction = System.Data.ParameterDirection.Input;

    keyParam.Value = date;

    var countParam = command.CreateParameter();

    countParam.ParameterName = "@Count";

    countParam.DbType = System.Data.DbType.Int32;

    countParam.Direction = System.Data.ParameterDirection.Input;

    countParam.Value = 100000;

    command.CommandType = System.Data.CommandType.Text;

    command.CommandText = string.Format(@"

```

```

USE {0};

```

```

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

```

```

DECLARE @End_Date DATETIME;

```

SELECT

 @End_Date = I.EndTime

FROM dbo.{1} AS I

WHERE I.EndTime > @Last_Date

ORDER BY

 I.EndTime

OFFSET (@Count - 1) ROWS FETCH NEXT (1) ROWS ONLY;

SELECT

(

 SELECT

 T.HostName,

 T.ApplicationName,

 T.Plexus_SQL_Server_Key,

 T.ServerName,

 T.Databaseld,

 T.Database_Name,

 T.SPID,

 T.EndTime,

 T.Duration,

 T.Reads,

 T.Writes,

 T.CPU,

 T.RowCounts,

 T.Error,


```

LN.Login_Name_Key,

LN.Login_Name,

T.textdata

FOR JSON PATH,

WITHOUT_ARRAY_WRAPPER

) AS JSON_Message,

T.EndTime

FROM Performance.dbo.{1} AS T

JOIN Performance.dbo.Login_Name AS LN

ON LN.Login_Name_Key = T.Login_Name_Key

WHERE T.EndTime > @Last_Date

AND T.EndTime <= @End_Date;",

```

```

    _database,

    _tableName);

```

```

command.Parameters.Add(keyParam);

command.Parameters.Add(countParam);

```

```

await connection.OpenAsync();

var reader = await command.ExecuteReaderAsync();

```

```

if (reader.HasRows)

{

    while (await reader.ReadAsync(cancellationToken))

    {

        transactions.Add(new TraceDataTable

```

```

        {

            JsonMessage = await Get<string>(reader, 0),

            EndTime = await Get<DateTime>(reader, 1)

        });

    }

}

connection.Close();

}

return transactions;

}

private async Task<T> Get<T>(SqlDataReader reader, int ordinal)

{

    var value = reader.GetValue(ordinal);

    try

    {

        return (await reader.IsDBNullAsync(ordinal)) ? default(T) : (T)reader.GetValue(ordinal);

    }

    catch (Exception ex)

    {

        return default(T);

    }

}

}

```

```
}
```

TraceDataTable.cs (Monitoring.ETL.Domain\Fact\SqlExecution\Trace\TraceDataTable.cs):

```
using ETL.Process;
```

```
using System;
```

```
namespace Monitoring.ETL.Domain.Fact.SqlExecution.Trace
```

```
{  
  
    public class TraceDataTable : IExtractModel, SqlJson.IJsonExtractModel  
  
    {  
  
        public DateTime EndTime { get; set; }  
  
        public string JsonMessage { get; set; }  
  
    }  
  
}
```

WarehouseLoader.cs (Monitoring.ETL.Domain\Fact\SqlExecution\WarehouseLoader.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.Fact.SqlExecution
```

```
{  
  
    public class WarehouseLoader : Loader<Warehouse.Model.SQL_Execution_w>  
  
    {  
  
        public WarehouseLoader()  
  
            : base("Fact_SQL_Execution_Add")  
  
        {
```

```
}  
  
}  
  
}
```

CustomerUxClassicClient.cs (Monitoring.ETL.Domain\HelperDataset\CustomerUxClassic\CustomerUxClassicClient.cs):

```
using ETL.Process;  
  
using Monitoring.ETL.Domain.ElasticSearch;  
  
using Monitoring.ETL.Domain.SqlJson;  
  
using Monitoring.ETL.Domain.Warehouse;  
  
using Monitoring.ETL.Process;  
  
using Monitoring.UserExtensions;  
  
using Newtonsoft.Json;  
  
using Newtonsoft.Json.Linq;  
  
using System;  
  
using System.Collections.Generic;  
  
using System.Data.SqlClient;  
  
using System.Linq;  
  
namespace Monitoring.ETL.Domain.HelperDataset.CustomerUxClassic  
{  
    /// <summary>  
    /// Retrieve a list of customers with a UX/Classic usage flag  
    /// </summary>
```

```

public class CustomerUxClassicClient : IHelperDataset
{
    private const string TrustedSqlServerConnectionStringTemplate = "data source={0};initial
catalog=Plexus_Control;integrated security=True;";

    private readonly ServiceToolConnectionRepository _serviceToolConnectionRepository = new
ServiceToolConnectionRepository();

    private readonly LoadStateRepository _state = new LoadStateRepository();

    public CustomerUxClassicClient(IEtlProcessLogger logger)
    {
        _logger = logger;
    }

    public IEtlProcessLogger _logger { get; set; }

    public ResultSet<T> AppendDataset<T>(ResultSet<T> resultSet) where T : IJsonExtractModel, new()
    {
        var results = new ResultSet<T>();

        foreach (var e in resultSet.Exceptions)
        {
            results.Exceptions.Add(e);
        }

        var customers = GetUxClassicCustomers();
    }
}

```

```
_logger.Information("Appending Ux/Classic information");

foreach (var record in resultSet.Results)

{

    var row = JsonConvert.DeserializeObject(record.JsonMessage) as JObject;

    if (row != null)

    {

        try

        {

            var pcn = Convert.ToInt32(row["PCN"]);

            var findCustomer = customers.SingleOrDefault(f => f.Pcn == pcn);

            if (findCustomer != null)

            {

                row["IsUx"] = findCustomer.IsUx;

                row["IsClassic"] = findCustomer.IsClassic;

            }

            else

            {

                row["IsUx"] = false;

                row["IsClassic"] = true; //default to classic if the PCN wasn't found.

            }

        }

        catch

        {

            //just ignore the error if the pcn isn't well formed for some reason

        }

    }

}
```

```
}
```

```
}
```

```
results.Results.Add(new T { JsonMessage = row.ToStringNull() });
```

```
}
```

```
return results;
```

```
}
```

```
public List<UxClassicCustomer> GetUxClassicCustomers()
```

```
{
```

```
//we can do this in one step, but this will be much easier to understand...
```

```
var uxClassicPercentage = GetUxClassicLoginPercent(); // Get ux/classic login percentage
```

```
var allCustomers = GetUxCustomers(); // get UX only. This is absolute.
```

```
//first, clear out all PCNs which are "UX Only" since we already have that list
```

```
uxClassicPercentage.RemoveAll(d => allCustomers.Select(f => f.Pcn).Contains(d.Pcn));
```

```
foreach (var c in uxClassicPercentage)
```

```
{
```

```
allCustomers.Add(new UxClassicCustomer
```

```
{
```

```
Pcn = c.Pcn,
```

```
IsUx = c.IsUx,
```

```
IsClassic = true
```

```
});
```

```
}
```

```
return allCustomers;
```

```
}
```

```
public List<UxClassicPercentage> GetUxClassicLoginPercent()
```

```
{
```

```
    _logger.Information("Retrieve UX/Classic login percentage");
```

```
    var query = @"declare @minDateTimeKey as int;
```

```
select @minDateTimeKey = dim_date_key from dbo.Datetime_To_Dim_Date_And_Time_Keys_Get(dateadd(d,-30,  
getdate()))
```

```
SELECT Pcn
```

```
,sum(classic) AS ClassicLogins
```

```
,sum(Cloud) AS CloudLogins
```

```
FROM (
```

```
SELECT c.pcn
```

```
,CASE
```

```
WHEN (left(login_origin, 5) = 'Class' OR left(login_origin, 5) = 'Cloud')
```

```
THEN sum(login_count)
```

```
ELSE 0
```

```
END AS TotalCount
```

```
,CASE
```

```
WHEN left(login_origin, 5) = 'Class'
```

```
THEN sum(l.login_count)
```

```
ELSE 0
```



```

END AS Classic

,CASE

WHEN left(login_origin, 5) = 'Cloud'

THEN sum(l.login_count)

ELSE 0

END AS Cloud

FROM fact_login l

INNER JOIN dim_customer c ON l.Dim_Customer_Key = c.Dim_Customer_Key

INNER JOIN Dim_Login_Origin lo ON l.Dim_Login_Origin_Key = lo.Dim_Login_Origin_Key

WHERE dim_date_key_login >= @minDateTimeKey

GROUP BY c.PCN

,lo.Login_Origin

) t

GROUP BY pcn

FOR JSON Path

";

```

```

        var warehouseRepo = new WarehouseRepository<UxClassicPercentage>(query,

@"dv_sqlperf.plex.com\sqlperf_prod");

        var results = warehouseRepo.ExecuteWarehouseQuery(true).Result;

        return results;

    }

```

```

/// <summary>

```

```

/// Get a list of UX Only customers. Run on each N

```

```

/// </summary>

```

```
/// <returns></returns>
```

```
public List<UxClassicCustomer> GetUxCustomers()
```

```
{
```

```
    _logger.Information("Retrieving UX Only customers");
```

```
    var customerList = new List<UxClassicCustomer>();
```

```
    const string query = @"SELECT pc.Plexus_customer_no
```

```
,pc.name
```

```
FROM plexus_control.dbo.Setting_Group AS SG
```

```
JOIN plexus_control.dbo.Setting AS S ON S.Setting_Group_Key = SG.Setting_Group_Key
```

```
AND S.Setting_Name = 'UX Only'
```

```
AND S.Setting_Type = 'Customer'
```

```
JOIN plexus_control.dbo.Plexus_Customer_Setting AS PCS ON PCS.Setting_Key = S.Setting_Key
```

```
JOIN plexus_control.dbo.Plexus_Customer AS PC ON PC.Plexus_Customer_No = pcs.Plexus_Customer_No
```

```
JOIN Plexus_System.dbo.Plexus_SQL_Server_Group AS PSSG ON PSSG.Plexus_SQL_Server_Group_Key =
```

```
PC.Plexus_SQL_Server_Group_Key
```

```
LEFT JOIN Plexus_Control.dbo.Customer_Group_Member AS CGM ON CGM.Plexus_Customer_No =
```

```
PC.Plexus_Customer_No
```

```
LEFT JOIN Plexus_Control.dbo.Customer_Group AS CG ON CGM.Customer_Group_No = CG.Customer_Group_No
```

```
WHERE PCS.Setting = 1
```

```
AND SG.Setting_Group = 'UX Transition'
```

```
AND PC.Plexus_Customer_Code NOT LIKE '%svcs%'
```

```
AND PC.Plexus_Customer_Code NOT LIKE '%PCN Move%'
```

```
AND PC.Plexus_Customer_Code NOT LIKE '%DEMO%'
```

```
AND PC.Plexus_Customer_Code NOT LIKE '%Template%'
```

```
AND PC.Plexus_Customer_Code NOT LIKE 'IAM%'
```

```
AND PC.Plexus_Customer_Code NOT LIKE 'PLEX %'
```

```
AND PC.Plexus_Customer_Code NOT LIKE '%GSS%'

AND PC.Plexus_Customer_Code NOT LIKE '%EDGE %'

AND PC.Plexus_Customer_Code NOT LIKE '%Utopia%'

AND PC.Plexus_Customer_Code NOT LIKE '%TSCH%'

AND PC.Plexus_Customer_Code NOT LIKE '%bgrant%'

AND PC.Plexus_Customer_Code NOT LIKE '%Data Center -%'

AND PC.Plexus_Customer_Code NOT LIKE '%Copy Process PCN%'

AND PC.Plexus_Customer_Code NOT LIKE '%Customer Care%'

AND PC.Plexus_Customer_Code NOT LIKE '%PowerPlex%'

AND PC.Plexus_Online_Customer = 1";
```

```
// even though we aren't technically the Login ETL, this list would always be the same.
```

```
var servers = _serviceToolConnectionRepository.GetServersToProcess("Login ETL").OrderBy(kvp => kvp.Key)

.ToList();
```

```
foreach (var server in servers)
```

```
{
```

```
    var key = server.Key.ToLower();
```

```
    key = key.Substring(key.IndexOf('n'));
```

```
    var underscore = key.IndexOf('_');
```

```
    if (underscore == -1)
```

```
        underscore = key.Length;
```

```
key = key.Substring(0, underscore).ToLower() + "_report";
```

```
_logger.Information(key);
```

```
var connectionString = Helpers.Util.GetSqlConnectionStringFromConsulId(key, "PLEXUS_CONTROL");
```

```
if (!string.IsNullOrEmpty(connectionString))
```

```
{
```

```
    using (var con = new SqlConnection(connectionString))
```

```
    {
```

```
        con.Open();
```

```
        using (var cmd = new SqlCommand(query, con))
```

```
        {
```

```
            using (var reader = cmd.ExecuteReader())
```

```
            {
```

```
                if (reader.HasRows)
```

```
                {
```

```
                    while (reader.Read())
```

```
                    {
```

```
                        customerList.Add(new UxClassicCustomer
```

```
                        {
```

```
                            Pcn = reader["Plexus_customer_no"].ToInt(),
```

```
                            IsUx = true
```

```
                        });
```

```
                    }
```

```
        }  
    }  
}  
  
}
```

```
    return customerList;  
}
```

```
public class UxClassicPercentage : IWarehouseLoadModel  
{  
    public int ClassicLoginPercent  
    {  
        get  
        {  
            if (TotalLogins == 0)  
                return 0;  
            return 100 - UxLoginPercent;  
        }  
    }  
}
```

```
public int ClassicLogins { get; set; }  
  
public int CloudLogins { get; set; }  
  
public bool IsUx => UxLoginPercent >= 30;  
  
public int Pcn { get; set; }
```

```
public int TotalLogins => ClassicLogins + CloudLogins;
```

```
public int UxLoginPercent
```

```
{
```

```
    get
```

```
    {
```

```
        if (TotalLogins == 0)
```

```
            return 0;
```

```
        return (CloudLogins * 100 / TotalLogins);
```

```
    }
```

```
}
```

```
/// <summary>
```

```
/// Totally not used here! Needed to satisfy IWarehouseLoadModel
```

```
/// </summary>
```

```
///
```

```
[JsonIgnore]
```

```
public int Warehouse_Load_Key { get; set; }
```

```
}
```

```
}
```

```
}
```

UxClassicCustomer.cs (Monitoring.ETL.Domain\HelperDataset\CustomerUxClassic\UxClassicCustomer.cs):

```
namespace Monitoring.ETL.Domain.HelperDataset.CustomerUxClassic
```

```
{
```

```
    public class UxClassicCustomer
```

```

{

    public bool IsClassic { get; set; }

    public bool IsUx { get; set; }

    public int Pcn { get; set; }

}

}

```

IHelperDataset.cs (Monitoring.ETL.Domain\HelperDataset\IHelperDataset.cs):

```

using System.Drawing.Text;

```

```

using ETL.Process;

```

```

using Monitoring.ETL.Domain.SqlJson;

```

```

using Monitoring.ETL.Process;

```

```

namespace Monitoring.ETL.Domain.HelperDataset

```

```

{

    /// <summary>

    /// An interface which describes functions for helper datasets

    /// </summary>

    internal interface IHelperDataset

    {

        IEtlProcessLogger _logger { get; set; }

        ResultSet<T> AppendDataset<T>(ResultSet<T> resultSet) where T : IJsonExtractModel, new();

    }

}

```

SalesforceClient.cs (Monitoring.ETL.Domain\HelperDataset\Salesforce\SalesforceClient.cs):

using System;

using System.Collections.Generic;

using Monitoring.UserExtensions;

using System.Configuration;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Net.Http.Headers;

using Newtonsoft.Json;

using Newtonsoft.Json.Linq;

using ETL.Process;

using Monitoring.ETL.Domain.SqlJson;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.HelperDataset.Salesforce

{

public class SalesforceClient : IHelperDataset

{

private const string CustomerQuery =

"SELECT id, name, Plex_Account_Number__c, EBC_Customer__c FROM Account where type='Customer'";

private string ApiEndpoint = "/services/data/v56.0/";


```

public SalesforceClient(string username, string password, string token, string clientid, string clientSecret)
{
    Username = username;

    Password = password;

    Token = token;

    ClientId = clientid;

    ClientSecret = clientSecret;
}

```

```

/// <summary>

```

```

/// Create a salesforce client with all pertinent settings

```

```

/// </summary>

```

```

public SalesforceClient(IEtlProcessLogger logger)

```

```

{
    _logger = logger;

```

```

    UrlEndpoint = ConfigurationManager.AppSettings["urlendpoint"].ToStringNull();

```

```

    if (string.IsNullOrEmpty(UrlEndpoint))

```

```

        throw new ArgumentNullException(nameof(UrlEndpoint), "Url Endpoint not set in application config");

```

```

    Username = ConfigurationManager.AppSettings["username"].ToStringNull();

```

```

    if (string.IsNullOrEmpty(Username))

```

```

        throw new ArgumentNullException(nameof(Username), "Username not set in application config");

```

```

    Password = ConfigurationManager.AppSettings["password"].ToStringNull();

```

```

    if (string.IsNullOrEmpty>Password))

```

```
throw new ArgumentNullException(nameof>Password, "Password not set in application config");
```

```
Token = ConfigurationManager.AppSettings["token"].ToStringNull();
```

```
if (string.IsNullOrEmpty(Token))
```

```
throw new ArgumentNullException(nameof(Token), "Token not set in application config");
```

```
ClientId = ConfigurationManager.AppSettings["clientId"].ToStringNull();
```

```
if (string.IsNullOrEmpty(ClientId))
```

```
throw new ArgumentNullException(nameof(ClientId), "ClientId not set in application config");
```

```
ClientSecret = ConfigurationManager.AppSettings["clientSecret"].ToStringNull();
```

```
if (string.IsNullOrEmpty(ClientSecret))
```

```
throw new ArgumentNullException(nameof(ClientSecret), "ClientSecret not set in application config");
```

```
ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12 | SecurityProtocolType.Tls11;
```

```
}
```

```
public string AuthToken { get; set; }
```

```
public string ClientId { get; set; }
```

```
public string ClientSecret { get; set; }
```

```
public string InstanceUrl { get; set; }
```

```
public string Password { get; set; }
```

```
public string Token { get; set; }
```

```
public string Username { get; set; }
```

```
public string LoginEndpoint => UrlEndpoint + "/services/oauth2/token";
```

```
public string UrlEndpoint { get; set; }
```

```
private string Describe(string sObject)
```

```
{  
  
    using (var client = new HttpClient())  
  
    {  
  
        var restQuery = InstanceUrl + ApiEndpoint + "subjects/" + sObject;  
  
        var request = new HttpRequestMessage(HttpMethod.Get, restQuery);  
  
        request.Headers.Add("Authorization", "Bearer " + AuthToken);  
  
        request.Headers.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));  
  
        request.Headers.Add("X-PrettyPrint", "1");  
  
        var response = client.SendAsync(request).Result;  
  
        return response.Content.ReadAsStringAsync().Result;  
  
    }  
}
```

```
private void Login()
```

```
{  
  
    string jsonResponse;  
  
    _logger.Information("Logging into Salesforce");  
  
    using (var client = new HttpClient())  
  
    {  
  
        var request = new FormUrlEncodedContent(new Dictionary<string, string>  
  
            {  
  
                {"grant_type", "password"},  
  
                {"client_id", ClientId},  

```

```

        {"client_secret", ClientSecret},

        {"username", Username},

        {"password", Password + Token}

    }

);

request.Headers.Add("X-PrettyPrint", "1");

var response = client.PostAsync(LoginEndpoint, request).Result;

jsonResponse = response.Content.ReadAsStringAsync().Result;

}

```

```

var values = JsonConvert.DeserializeObject<Dictionary<string, string>>(jsonResponse);

AuthToken = values["access_token"];

InstanceUrl = values["instance_url"];

}

```

/// <summary>

/// Execute a query and return the JToken

/// </summary>

/// <param name="soqlQuery"></param>

/// <returns></returns>

private JObject Query(string soqlQuery)

```

{

    _logger.Information("Executing Salesforce Query");

    using (var client = new HttpClient())

    {

        var restRequest = InstanceUrl + ApiEndpoint + "query/?q=" + soqlQuery;

```

```

var request = new HttpRequestMessage(HttpMethod.Get, restRequest);

request.Headers.Add("Authorization", "Bearer " + AuthToken);

request.Headers.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

request.Headers.Add("X-PrettyPrint", "1");

var response = client.SendAsync(request).Result;

var jsonResult = response.Content.ReadAsStringAsync().Result;

var d = JObject.Parse(jsonResult);

return d;
}
}

```

```

private string QueryEndpoints()
{
    using (var client = new HttpClient())
    {
        var restQuery = InstanceUrl + ApiEndpoint;

        var request = new HttpRequestMessage(HttpMethod.Get, restQuery);

        request.Headers.Add("Authorization", "Bearer " + AuthToken);

        request.Headers.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));

        request.Headers.Add("X-PrettyPrint", "1");

        var response = client.SendAsync(request).Result;

        return response.Content.ReadAsStringAsync().Result;
    }
}

```

```

private List<SalesforceCustomer> GetCustomers()

{

    var jDocument = Query(CustomerQuery);

    var jsonRecords = jDocument["records"];

    var list = new List<SalesforceCustomer>();

    if (jsonRecords != null)

    {

        _logger.Information($"{jsonRecords.Count()} Salesforce records retrieved");

        foreach (var p in jsonRecords)

        {

            var pcn = p["Plex_Account_Number__c"].ToInt();

            if (pcn > 0)

            {

                var item = new SalesforceCustomer

                {

                    Pcn = pcn,

                    EbcCustomer = p["EBC_Customer__c"].ToStringNull().ToBool(),

                    SalesforceId = p["Id"].ToStringNull(),

                    CustomerName = p["Name"].ToStringNull() ?? ""

                };

                list.Add(item);

            }

        }

    }

}

```

```

    }

}

else

{

    _logger.Information("No customer information retrieved from Salesforce");

}


return list;


}


public IEtlProcessLogger _logger { get; set; }


public ResultSet<T> AppendDataset<T>(ResultSet<T> resultSet) where T : IJsonExtractModel, new()
{

    var results = new ResultSet<T>();


    foreach (var e in resultSet.Exceptions)

        results.Exceptions.Add(e);


    Login();


    var customers = GetCustomers();


    foreach (var record in resultSet.Results)

    {

```

```
var row = JsonConvert.DeserializeObject(record.JsonMessage) as JObject;
```

```
if (row != null)
```

```
{
```

```
    try
```

```
    {
```

```
        var pcn = Convert.ToInt32(row["PCN"]);
```

```
        var findPcn = customers.FirstOrDefault(f => f.Pcn == pcn);
```

```
        if (findPcn != null)
```

```
        {
```

```
            row["IsEbc"] = findPcn.EbcCustomer;
```

```
        }
```

```
    else
```

```
    {
```

```
        row["IsEbc"] = false;
```

```
    }
```

```
}
```

```
catch
```

```
{
```

```
    row["IsEbc"] = false;
```

```
    //just ignore the error if the pcn isn't well formed for some reason and mark EBC as false
```

```
}
```

```
}
```

```
results.Results.Add(new T { JsonMessage = row.ToStringNull() });
```



```
}
```

```
return results;
```

```
}
```

```
}
```

```
}
```

SalesforceCustomer.cs (Monitoring.ETL.Domain\HelperDataset\Salesforce\SalesforceCustomer.cs):

```
namespace Monitoring.ETL.Domain.HelperDataset.Salesforce
```

```
{
```

```
    public class SalesforceCustomer
```

```
    {
```

```
        public bool EbcCustomer { get; set; }
```

```
        public string SalesforceId { get; set; }
```

```
        public string CustomerName { get; set; }
```

```
        public int Pcn { get; set; }
```

```
    }
```

```
}
```

ShardClient.cs (Monitoring.ETL.Domain\HelperDataset\Shard\ShardClient.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.Customer;
```

```
using Monitoring.ETL.Domain.SqlJson;
```

```
using Newtonsoft.Json;
```

```
using Newtonsoft.Json.Linq;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.IO;
```

```
using System.Linq;
```

```
using System.Net;
```

```
using Monitoring.ETL.Process;
```

```
using Monitoring.UserExtensions;
```

```
namespace Monitoring.ETL.Domain.HelperDataset.Shard
```

```
{
```

```
    /// <summary>
```

```
    /// Client for retrieving shard information
```

```
    /// </summary>
```

```
    public class ShardClient : IHelperDataset
```

```
    {
```

```
        private const string ShardUrl = "http://dv-nscale-p01/v1/info";
```

```
        public ShardClient(IETLProcessLogger logger) { _logger = logger; }
```

```
        public IETLProcessLogger _logger { get; set; }
```

```

/// <summary>

/// Append the shard info to the current result set. This only works if PCN is

/// available in the data set.

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="resultSet"></param>

/// <returns></returns>

public ResultSet<T> AppendDataset<T>(ResultSet<T> resultSet) where T : IJsonExtractModel, new()
{
    var results = new ResultSet<T>();

    _logger.Information("Adding Exceptions");

    foreach (var e in resultSet.Exceptions)
    {
        results.Exceptions.Add(e);
    }

    _logger.Information("Retrieving Shard information");

    var shardInfo = GetShardInfo();

    _logger.Information("Appending Shard information");

    foreach (var record in resultSet.Results)
    {
        var row = JsonConvert.DeserializeObject(record.JsonMessage) as JObject;

        if (row != null)
        {
            try
            {
                {

```

```
var pcn = Convert.ToInt32(row["PCN"]);
```

```
var findPcn = shardInfo.SingleOrDefault(d => d.Id == pcn);
```

```
if (findPcn != null)
```

```
{
```

```
    row["Shard"] = findPcn.Shard.ToUpper();
```

```
}
```

```
}
```

```
catch
```

```
{
```

```
    //just ignore the error if the pcn isn't well formed for some reason
```

```
}
```

```
}
```

```
results.Results.Add(new T { JsonMessage = row.ToStringNull() });
```

```
}
```

```
return results;
```

```
}
```

```
/// <summary>
```

```
/// Get shard information
```

```
/// </summary>
```

```
/// <returns></returns>
```

```
public List<CustomerShard> GetShardInfo()
```

```
{
```

```

var request = (HttpWebRequest)WebRequest.Create(ShardUrl);

request.Method = "GET";

request.ContentType = "application/json";

var response = request.GetResponse();

var stream = response.GetResponseStream();


if (stream != null)
{
    var responseString = new StreamReader(stream).ReadToEnd();

    var shardInfo = JsonConvert.DeserializeObject<List<CustomerShard>>(responseString);

    return shardInfo;
}

return null;
}
}
}

```

Util.cs (Monitoring.ETL.Domain\Helpers\Util.cs):

```

using System;

using System.Linq;

using System.Xml.Linq;


using Monitoring.ETL.Process;


namespace Monitoring.ETL.Domain.Helpers

```

```

{

    public static class Util

    {

        public static string GetSqlConnectionStringFromConsulId(string consulId, string dataBase, IEtlProcessLogger
logger = null)

        {

            var consulTemplate = XDocument.Load(@"c:\infrastructure.xml");

            var node = (from c in consulTemplate.Root?.Element("services")?
                        .Element("databases")?.Elements("service")
                        where (string)c.Attribute("id") == consulId
                        select c).SingleOrDefault();

            string connectionString;

            if (node != null)
            {

                var host = node.Attribute("host")?.Value;

                var port = node.Attribute("port")?.Value;

                var username = node.Attribute("username")?.Value;

                var password = node.Attribute("password")?.Value;

                connectionString = string.Format(

                    username != null && password != null ?

                        "data source={0},{1};initial catalog={4};user={2};password={3};" :

                        "data source={0},{1};initial catalog={4};Integrated Security=true;",

```

```

        host,

        port,

        username,

        password,

        dataBase);

    }

    else

    {

        logger?.Information($"@\"c:\infrastructure.xml is not found or service ID of '{consulId}' does not exist");

        return "";

    }

    return connectionString;

}

}

}

```

JiraIssue.cs (Monitoring.ETL.Domain\Jira\JiraIssue.cs):

```

using ETL.Process;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Monitoring.ETL.Domain.Jira

{

```

```
/// <summary>
```

```
/// A model which represents a jira issue
```

```
/// </summary>
```

```
public class JiraIssue : IExtractModel
```

```
{
```

```
    public string JsonMessage { get; set; }
```

```
}
```

```
}
```

JiraIssueDelayFactory.cs (Monitoring.ETL.Domain\Jira\JiraIssueDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Jira
```

```
{
```

```
    /// <summary>
```

```
    /// Defines how often the JiraIssue runs
```

```
    /// </summary>
```

```
public sealed class JiraIssueDelayFactory : ThresholdExtractionDelayFactory<JiraIssue>
```

```
{
```

```
    /// <summary>
```

```
    /// Matches the number of records we're pulling back at one time from the JiraIssueExtrator
```

```
    /// </summary>
```

```
protected override int MaxThresholdForDelay
```

```
{
```

```
    get
```

```
{
```



```
        int threshold = 1;

        return threshold;

    }

}
```

```
protected override TimeSpan DelayTimeSpan

{

    get

    {

        int seconds = 60;

        return TimeSpan.FromSeconds(seconds);

    }

}

}
```

JiraIssueExtractor.cs (Monitoring.ETL.Domain\Jira\JiraIssueExtractor.cs):

```
using ETL.Process;

using Monitoring.Email.Models;

using Monitoring.Email.Services;

using Monitoring.ETL.Domain.ElasticSearch;

using Monitoring.ETL.Process;

using Newtonsoft.Json;

using Newtonsoft.Json.Linq;

using System;

using System.Collections.Generic;
```

```
using System.Configuration;
```

```
using System.Linq;
```

```
using System.Net.Http;
```

```
using System.Net.Http.Headers;
```

```
using System.Text;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.Jira
```

```
{
```

```
    /// <summary>
```

```
    /// Extract from Jira using the Jira API and returns the result set
```

```
    /// </summary>
```

```
    public class JiraIssueExtractor : IExtractor<JiraIssue>
```

```
    {
```

```
        private EmailService services = new EmailService();
```

```
        private readonly LoadStateRepository<JiraIssue> _loadStateRepository;
```

```
        private DateTime _lastDate;
```

```
        private string JiraUrl = "https://plexsystems.atlassian.net";
```

```
        private int maxResults;
```

```
        private int numMinutesMinimum = 60 * 24 * 7; //one week default
```

```
        private ResultSet<JiraIssue> resultSet;
```

```
        private DateTime firstJiraTicket = new DateTime(2015, 5, 29, 12, 0, 0); //the first ticket in the jira system, start at
```

```
noon on 5/29/15
```

//id 116818 is a special epam ticket with currently 16000+ changelog records

```
private string JiraInitialSearch = "{0}/rest/api/2/search?startAt=0&fields=id&maxResults=1000&jql=updated>='{1}'
```

and updated<='{2}' and project!=10800 order by updated ASC";

```
private string JiraFinalSearch = "{0}/rest/api/2/search?startAt=0&maxResults=1000&expand=changelog&jql=id in
```

(({1}) order by id ASC";

//Note: previously stored credentials in the repo have been disabled.

```
private string jiraUserId = ConfigurationManager.AppSettings["JiraUserId"];
```

```
private string jiraPassword = ConfigurationManager.AppSettings["JiraPassword"];
```

//jira search is in local time, but the results come back UTC. This will convert the UTC datetime to local

automatically.

```
private JsonSerializerSettings jsonSerializerSettings = new JsonSerializerSettings { DateTimeZoneHandling =
```

```
DateTimeZoneHandling.Local };
```

```
private DateTime? startDate, endDate;
```

//e.g. 2019-01-01 14:52

//as of 11/27/2019, jira does not include support for ordering or filtering by seconds. Seconds are COMPLETELY ignored. For example,

//if the

```
private string jiraDateTimeFormat = "{0:00}-{1:00}-{2:00} {3:00}:{4:00}";
```

```
public JiraIssueExtractor()
```

```
{
```

```

int.TryParse(ConfigurationManager.AppSettings["Jira_Max_Number_Issues_Retrieved"], out maxResults);

if (maxResults == 0)

    maxResults = 20;


if (startDate == null)

    startDate = DateTime.TryParse(ConfigurationManager.AppSettings["Jira_Start_Date"], out DateTime sd) ? sd
: (DateTime?)null;


if (endDate == null)

    endDate = DateTime.TryParse(ConfigurationManager.AppSettings["Jira_End_Date"], out DateTime ed) ? ed :
(DateTime?)null;


if (startDate != null)

    _lastDate = (DateTime)startDate;


_loadStateRepository = new LoadStateRepository<JiraIssue>();
}


private void LogInformation(IEtlProcessLogger logger, string message)
{
    if (logger != null)

        logger.Information(message);
}


public async Task<ResultSet<JiraIssue>> ExtractAllSinceLastExtractAsync(Cancellation_token cancellationToken,
IEtlProcessLogger logger)

```

```

{

    DateTime minimumDateDt = default, maximumDateDt = default;

    resultSet = new ResultSet<JiraIssue>();

    ChangedJiraIds jiraIds = null;

    using (HttpClient httpClient = new HttpClient())

    {

        try

        {

            if (_lastDate == default)

            {

                _lastDate = _loadStateRepository.GetDate();

                if (_lastDate < firstJiraTicket)

                {

                    _lastDate = firstJiraTicket;

                }

            }

            minimumDateDt = _lastDate.AddSeconds(-_lastDate.Second); //strip the seconds

            //if we specify a hard end date, then exit once we're done.

            if (minimumDateDt >= endDate)

                return resultSet;

            //don't process a partial day

            if (minimumDateDt.Date >= DateTime.Now.Date)

            {

                _lastDate = default(DateTime);

            }

        }

    }
}

```

```

        return resultSet;
    }

    //if the date is within a week, pull back 1 day at a time

    if (minimumDateDt.Date.AddDays(7) >= DateTime.Now.Date)

        numMinutesMinimum = 1440;

    //if the date is within a day, pull back 1 hour at a time

    if (minimumDateDt.Date.AddDays(1) == DateTime.Now.Date)

        numMinutesMinimum = 60;

    if (minimumDateDt.Year < 1970)

        minimumDateDt = firstJiraTicket;

    maximumDateDt = minimumDateDt.AddMinutes(numMinutesMinimum);

    // will make it so we don't pull in any records for the current date

    if (maximumDateDt.Date >= DateTime.Now.Date)

        maximumDateDt = DateTime.Now.Date;

    if (endDate != null && maximumDateDt > endDate)

        maximumDateDt = (DateTime)endDate;

    LogInformation(logger, "Finding Jira Ids which have updated date >=" + minimumDateDt.ToString() + " & <"
+ maximumDateDt.ToString());

```

```

var totalIds = 99999;

//We can't pull back more than 1000 at a time, so if the total is 1000, then reduce the timeframe in half

while (totalIds >= 1000)

{

    jiralds = GetJiraldsWhichHaveChanged(minimumDateDt, maximumDateDt, logger);

    totalIds = jiralds.TotalIssues;

    if (jiralds?.TotalIssues >= 1000)

    {

        TimeSpan ts = maximumDateDt.Subtract(minimumDateDt);

        ts = new TimeSpan(ts.Ticks / 2);

        maximumDateDt = minimumDateDt.Add(ts);

    }

}

if (jiralds?.TotalIssues > 0)

    resultSet = LoadJiralIssues(jiralds.Jiralds, logger);

if (maximumDateDt > _lastDate && jiralds != null && (jiralds.Jiralds != null || jiralds.TotalIssues >= 0))

{

    _lastDate = maximumDateDt;

    await _loadStateRepository.SetDate(maximumDateDt);

    await _loadStateRepository.SetKey(0);

}

}

```

```
catch (Exception ex)
```

```
{
```

```
    LogAndEmailException(ex);
```

```
    //we won't update the last date if there is an error. This will force a reload of the data from the last load
```

```
date.
```

```
    minimumDateDt = _lastDate;
```

```
}
```

```
    return resultSet;
```

```
}
```

```
}
```

```
public Task<IEnumerable<JiraIssue>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
```

```
CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
{
```

```
    throw new NotImplementedException();
```

```
}
```

```
private void LogAndEmailException(Exception ex)
```

```
{
```

```
    if (resultSet != null)
```

```
        resultSet.Exceptions.Add(ex);
```

```
    EmailViewModel email = new EmailViewModel();
```

```
    SMTPServerViewModel smtpServer = new SMTPServerViewModel
```



```

{
    EnableSsl = true,

    SMTPPort = Convert.ToInt32(ConfigurationManager.AppSettings["smtpPort"]),

    Password = ConfigurationManager.AppSettings["smtpPassword"],

    Username = ConfigurationManager.AppSettings["smtpUsername"],

    SMTPServer = ConfigurationManager.AppSettings["smtpServer"]
};

email.FromName = "Monitoring ETL Services";

email.FromEmail = "monitordevs@plex.com";

email.RecipientString = "monitordevs@plex.com";

email.Message = "An exception occurred in the Monitoring.ETL.Domain for JiraIssueExtrator: <br/><br/>";

email.Message += ex.Message;

email.EmailType = Email.Abstract Enumeration.EmailType.NoLog;

email.Subject = "Monitoring ETL Services - JiraIssueExtrator.cs";

if (ex.InnerException != null)

    email.Message += ex.InnerException.ToString() + "<br/><br/>";

if (ex.StackTrace != null)

    email.Message += ex.StackTrace.ToString();

var result = services.SendMail(email, smtpServer);
}

/// <summary>

/// Get a list of jira internal ids which have changed between two specific dates

```

```
/// </summary>
```

```
/// <param name="startdate"></param>
```

```
/// <param name="enddate"></param>
```

```
/// <param name="logger"></param>
```

```
/// <returns></returns>
```

```
public ChangedJirals GetJiralsWhichHaveChanged(DateTime startdate, DateTime enddate, IEtlProcessLogger  
logger)
```

```
{
```

```
    ChangedJirals ids = null;
```

```
    List<int> jirals = new List<int>();
```

```
    using (HttpClient httpClient = new HttpClient())
```

```
{
```

```
    try
```

```
{
```

```
        string minimumDate = string.Format(jiraDateTimeFormat, startdate.Year, startdate.Month, startdate.Day,  
startdate.Hour, startdate.Minute);
```

```
        string maximumDate = string.Format(jiraDateTimeFormat, enddate.Year, enddate.Month, enddate.Day,  
enddate.Hour, enddate.Minute);
```

```
        httpClient.Timeout = new TimeSpan(0, 10, 0); //set a 10 min timeout. This should be PLENTY
```

```
        var byteArray = Encoding.ASCII.GetBytes(string.Format("{0}:{1}", jiraUserId, jiraPassword));
```

```
        var header = new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));
```

```
        httpClient.DefaultRequestHeaders.Authorization = header;
```

```
LogInformation(logger, "Finding Issues >=" + minimumDate + " and <" + maximumDate);
```

```
string requestUrl = string.Format(JiraInitialSearch, JiraUrl, minimumDate, maximumDate);
```

```
string initialResponse = httpClient.GetStringAsync(requestUrl).Result;
```

```
try
```

```
{
```

```
    ids = new ChangedJiraIds();
```

```
    var myObject = JsonConvert.DeserializeObject<JObject>(initialResponse, jsonSerializerSettings);
```

```
    ids.TotalIssues = (int)myObject["total"];
```

```
    LogInformation(logger, "Total Issues Found: " + ids.TotalIssues.ToString());
```

```
    //if we have issues, find all jira ids
```

```
    if (ids.TotalIssues > 0 && ids.TotalIssues < 1000)
```

```
    {
```

```
        for (int i = 0; i < ids.TotalIssues; i++)
```

```
        {
```

```
            var jiraId = (int)myObject["issues"][i]["id"];
```

```
            jiraIds.Add(jiraId);
```

```
        }
```

```
        ids.JiraIds = jiraIds.OrderBy(d => d).Distinct().ToList();
```

```
    }
```

```
    else
```

```
    {
```

```
        if (ids.TotalIssues >= 1000)
```

```

        {
            LogInformation(logger, "Issue Count >=1000, reducing time range.");
        }
    }
}

catch (Exception ex)
{
    LogAndEmailException(ex);
}

}

catch (Exception ex)
{
    LogAndEmailException(ex);
}

return ids;
}
}

```

```

public ResultSet<JiraIssue> LoadJiraIssues(List<int> jiralds, IEtlProcessLogger logger)

```

```

{

```

///jira search is in local time, but the results come back UTC. This will convert the UTC datetime to local automatically.

```

        JsonSerializerSettings jsonSerializerSettings = new JsonSerializerSettings { DateTimeZoneHandling =
        DateTimeZoneHandling.Local };
    
```

```

var resultSet = new ResultSet<JiraIssue>(); //load this

int startAt = 0;

int totalIssues = jiraIds.Count();

using (HttpClient httpClient = new HttpClient())
{
    httpClient.Timeout = new TimeSpan(0, 10, 0); //set a 10 min timeout. This should be PLENTY

    var byteArray = Encoding.ASCII.GetBytes(string.Format("{0}:{1}", jiraUserId, jiraPassword));

    var header = new AuthenticationHeaderValue("Basic", Convert.ToBase64String(byteArray));

    httpClient.DefaultRequestHeaders.Authorization = header;

    try
    {
        while (startAt < totalIssues)
        {
            var jiraSubset = jiraIds.Skip(startAt).Take(maxResults);

            string jql = string.Join(",", jiraSubset);

            string requestUrl = string.Format(JiraFinalSearch, JiraUrl, jql);

            //we are doing this deserialize/serialize dance so we don't have to manually parse the "issues" element

            string finalResponse = httpClient.GetStringAsync(requestUrl).Result;

            var finalObject = JsonConvert.DeserializeObject<JObject>(finalResponse, jsonSerializerSettings);

            var totalItems = (int)finalObject["total"];

```

```

        LogInformation(logger, "Retrieving " + totalIssues + " issues, startAt=" + startAt.ToString() + "
maxResults=" + maxResults.ToString());

        if (finalObject["issues"].Count() > 0)
        {
            for (int issue = 0; issue < finalObject["issues"].Count(); issue++)
            {
                var sissue = JsonConvert.SerializeObject(finalObject["issues"][issue]);

                resultSet.Results.Add(new JiraIssue { JsonMessage = sissue });
            }
        }

        startAt += maxResults;
    }
}

catch (Exception ex)
{
    LogAndEmailException(ex);
}

return resultSet;
}
}

public class ChangedJirals
{

```

```

        public int TotalIssues;

        public List<int> Jiralds;

    }

}

}

```

JiralIssueWarehouseLoader.cs (Monitoring.ETL.Domain\Jira\JiralIssueWarehouseLoader.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.Jira
```

```

{

    public class JiralIssueWarehouseLoader : Loader<Warehouse.Model.Jira_Issues_w>

    {

        public JiralIssueWarehouseLoader() : base("Fact_Jira_Issue_Add")

        {

        }

    }

}

```

RabbitMQDelayFactory.cs (Monitoring.ETL.Domain\Jira\RabbitMQDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Jira
```

```

{

    /// <summary>

    /// The amount of delay to load extract rabbit mq data.

```

/// </summary>

```
public class RabbitMQDelayFactory : ThresholdExtractionDelayFactory<RabbitMQ.Model.RabbitMQJiraIssue>

{

    protected override int MaxThresholdForDelay => int.MaxValue;


    protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);

}

}
```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\Jira\RabbitMQExtractor.cs):

```
using System;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;


namespace Monitoring.ETL.Domain.Jira

{

    public sealed class RabbitMQExtractor : IExtractor<RabbitMQ.Model.RabbitMQJiraIssue>

    {

        private RabbitMQ.Jira.Consumer _consumer;


        public async Task<ResultSet<RabbitMQ.Model.RabbitMQJiraIssue>>

ExtractAllSinceLastExtractAsync(CancellationToken cancellationToken, IEtlProcessLogger logger)

        {
```



```

        _consumer = _consumer ?? new RabbitMQ.Jira.Consumer(logger);

        return await _consumer.ExtractAsync();
    }

```

```

    public Task<IEnumerable<RabbitMQ.Model.RabbitMQJiraIssue>> ExtractBetweenAsync(DateTime startDate,
    DateTime endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)
    {
        throw new NotImplementedException();
    }
}

```

RabbitMQJiraIssueLoader.cs (Monitoring.ETL.Domain\Jira\RabbitMQJiraIssueLoader.cs):

```

using Monitoring.ETL.Domain.Warehouse;

```

```

namespace Monitoring.ETL.Domain.Jira

```

```

{
    public sealed class RabbitMQJiraIssueLoader : RabbitMQ.Loader<RabbitMQ.Model.RabbitMQJiraIssue>
    {
        public RabbitMQJiraIssueLoader() : base(new RabbitMQ.Jira.Producer())
        {
        }
    }
}

```

RabbitMQJiraIssueTransformer.cs (Monitoring.ETL.Domain\Jira\RabbitMQJiraIssueTransformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.Login.Extract;
```

```
using Monitoring.ETL.Domain.Login.Load;
```

```
using Monitoring.ETL.Domain.RabbitMQ.Model;
```

```
using Monitoring.ETL.Domain.Warehouse.Model;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Jira
```

```
{
```

```
    public sealed class JiraIssueTransformer : ITransformer<JiraIssue, RabbitMQJiraIssue>
```

```
    {
```

```
        /// <summary>
```

```
        /// Transform the JiraIssue to the RabbitMQJiraIssue. In this case the models are essentially the same.
```

```
        /// </summary>
```

```
        /// <param name="extracted"></param>
```

```
        /// <param name="cancellationToken"></param>
```

```
        /// <param name="logger"></param>
```

```
        /// <returns></returns>
```

```
        public async Task<IEnumerable<RabbitMQJiraIssue>> TransformAsync(IEnumerable<JiraIssue> extracted,
```

```
CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
    {
```

```
        var issues = extracted.Select(d => new RabbitMQJiraIssue { JsonMessage = d.JsonMessage });
```

```

        return issues;
    }

}

```

RabbitMQWarehouseTransformer.cs (Monitoring.ETL.Domain\Jira\RabbitMQWarehouseTransformer.cs):

```

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;


namespace Monitoring.ETL.Domain.Jira
{
    /// <summary>

    /// Transform the rabbit mq message into the warehouse. The models are essentially the same with Jira, being Json
of the Jira issues.

    /// </summary>

    public class RabbitMQWarehouseTransformer : ITransformer<RabbitMQ.Model.RabbitMQJiraIssue,
Warehouse.Model.Jira_Issues_w>

    {

        public async Task<IEnumerable<Warehouse.Model.Jira_Issues_w>>

TransformAsync(IEnumerable<RabbitMQ.Model.RabbitMQJiraIssue> extracted, CancellationToken cancellationToken,
IEtlProcessLogger logger)

```

```

{
    await Task.Yield();

    return extracted.Select(pc =>
        new Warehouse.Model.Jira_Issues_w
        {
            JSON_Message = pc.JsonMessage
        });
    }
}
}

```

DimRepository.cs (Monitoring.ETL.Domain\Kpis\DimRepository.cs):

```

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Reflection;

using System.Threading.Tasks;

```

```

namespace Monitoring.ETL.Domain.Kpis
{
    public class DimRepository : IDimRepository
    {

```

```
private readonly string _connectionString;
```

```
public DimRepository(string connectionString)
```

```
{
```

```
    _connectionString = connectionString;
```

```
}
```

```
public async Task<List<T>> LoadAll<T>(Func<T, bool> filter = null)
```

```
{
```

```
    List<T> results = new List<T>();
```

```
    string sql = GetLoadSql<T>();
```

```
    using (var connection = new SqlConnection(_connectionString))
```

```
    {
```

```
        await connection.OpenAsync();
```

```
        using (var command = new SqlCommand(sql, connection))
```

```
        {
```

```
            command.CommandType = CommandType.Text;
```

```
            var reader = await command.ExecuteReaderAsync();
```

```
            while (await reader.ReadAsync())
```

```
            {
```

```
                results.Add(LoadFromReader<T>(reader));
```

```
            }
```

```
        }
```

```
    }
```

// At some point we might want to pre-filter the SQL, but currently the data returned is small enough the code isn't

worth it

```
return filter == null ? results : results.Where(filter).ToList();
```

```
}
```

```
private string GetLoadSql<T>()
```

```
{
```

```
return $"SELECT * FROM [dbo].[{GetTableName<T>()}]";
```

```
}
```

```
private string GetTableName<T>()
```

```
{
```

```
var att = typeof(T).GetCustomAttribute<TableAttribute>();
```

```
if (att == null)
```

```
{
```

```
throw new MissingFieldException($"{typeof(T).Name} doesn't have a {typeof(TableAttribute).Name}");
```

```
}
```

```
return att.Name;
```

```
}
```

```
private T LoadFromReader<T>(SqlDataReader reader)
```

```
{
```

```
T output = Activator.CreateInstance<T>();
```

```
for (int i = 0; i < reader.FieldCount; i++)
```

```

{

    var pi = typeof(T).GetProperty(reader.GetName(i));

    object value = reader.GetValue(i);

    value = value == DBNull.Value ? null : value;

    pi.SetValue(output, value);

}

return output;

}

}

}

```

IDimRepository.cs (Monitoring.ETL.Domain\Kpis\IDimRepository.cs):

```

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace Monitoring.ETL.Domain.Kpis

{

    public interface IDimRepository

    {

        Task<List<T>> LoadAll<T>(Func<T, bool> filter = null);

    }

}

```

NameValueCollectionExtensions.cs (Monitoring.ETL.Domain\Kpis\NameValueCollectionExtensions.cs):

```

using System;

```

```
using System.Collections.Specialized;
```

```
using System.Configuration;
```

```
using Monitoring.ETL.Domain.Kpis.Source.Models;
```

```
namespace Monitoring.ETL.Domain.Kpis
```

```
{
```

```
    public static class NameValueCollectionExtensions
```

```
    {
```

```
        /// <summary>
```

```
        /// Helper method for getting a configuration value with a default fallback options
```

```
        /// </summary>
```

```
        /// <param name="collection">The current <see cref="NameValueCollection"/></param>
```

```
        /// <param name="name">The name of the value to be retrieved</param>
```

```
        /// <param name="defaultValue">The default value if the collection either doesn't have that name defined or the value  
is null.</param>
```

```
        /// <returns>The value from the collection or the default value</returns>
```

```
        public static string GetValueOrDefault(this NameValueCollection collection, string name, string defaultValue)
```

```
        {
```

```
            return collection[name] ?? defaultValue;
```

```
        }
```

```
        /// <summary>
```

```
        /// Helper method for getting a configuration value with a default fallback options
```

```
        /// </summary>
```

```
        /// <param name="collection">The current <see cref="NameValueCollection"/></param>
```

```
        /// <param name="name">The name of the value to be retrieved</param>
```


/// <param name="defaultValue">The default value if the collection either doesn't have that name defined or the value is null.</param>

/// <returns>The value from the collection or the default value</returns>

```
public static int GetValueOrDefault(this NameValueCollection collection, string name, int defaultValue)
```

```
{  
  
    return int.TryParse(collection[name], out int value) ? value : defaultValue;  
  
}
```

/// <summary>

/// Helper method for getting a configuration value with a default fallback options

/// </summary>

/// <param name="collection">The current <see cref="NameValueCollection"/></param>

/// <param name="name">The name of the value to be retrieved</param>

/// <param name="defaultValue">The default value if the collection either doesn't have that name defined or the value is null.</param>

/// <returns>The value from the collection or the default value</returns>

```
public static bool GetValueOrDefault(this NameValueCollection collection, string name, bool defaultValue)
```

```
{  
  
    return bool.TryParse(collection[name], out bool value) ? value : defaultValue;  
  
}
```

/// <summary>

/// Helper method for getting a configuration value with a default fallback options

/// </summary>

/// <param name="collection">The current <see cref="NameValueCollection"/></param>

/// <param name="name">The name of the value to be retrieved</param>

```
/// <param name="defaultValue">The default value if the collection either doesn't have that name defined or the value
is null.</param>
```

```
/// <returns>The value from the collection or the default value</returns>
```

```
public static T GetValueOrDefault<T>(this NameValueCollection collection, string name, T defaultValue) where T :
struct
{
    return Enum.TryParse<T>(collection[name], true, out T value) ? value : defaultValue;
}
}
}
```

ReadMe.md (Monitoring.ETL.Domain\Kpis\ReadMe.md):

AppDev KPI Capture

This section of the Meta project is focused on capturing raw metrics from various sources that can be used to compile KPI metrics for the UX applications. There are a few slight differences between how this section works, thus this document.

Configuration

There are several configuration values that may be required to run the extration. These values can be stored in the `AppSettings` section of `App.Config`.

Key	Required	Description
-----	----------	-------------

---	---	---
-----	-----	-----

Jira:Url	No	The URL to the Jira server. This value defaults to https://plexsystems.atlassian.net , but can be overridden via configuration.
----------	----	---

| Jira:Username | Yes | The user id used to authenticate to Jira.|

| Jira:Token | Yes | The access key used to authenticate to Jira.|

| SonarQube:Url |No | The URL to the SonarQube server. This value defaults to https://sonarqube.aze.plex, but can be overridden via configuration|

| SonarQube:Token| Yes | The access key used to authenticate to SonarQube.|

| Kpis:RunNow | No | Indicates that the KPI extraction should take place immediately. The default value is `false` but it can be set to `true` for debugging. There is no reason for this to be set in production.|

| Kpis:NextRun | No | Indicates the timestamp the next KPI extraction should be run. If this date is set in the past, it is ignored. This is primarily used to restart the extraction interval if necessary. The date should be in the following format `yyyy-MM-ddTHH:mm:ss`.|

|Kpis:BuildDay| No | Indicates which day of the week releases are made. This defaults to `Tuesday` but is available if our normal build day changes. We want to run the exctraction on a release day.|

|Kpis:BuidIntervalWeeks| No | Indicates how many weeks there are between releases. This defaults to `2`. We can change this if we adjust our sprint length. The only values currently supported are `1`, `2`, and `3`. Illegal values will use the default value.|

|Kpis:RunHour| No | Indicates which hour during the day to run the extraction. This defaults to `12` as we want to run at noon on release day. Values can be set between `0` and `23`. Illegal values will use the default value.|

Several extension methods were added to handle optiional configuration values. These can be found in

`NameValueCollectionExtensions` under the method `GetValueOrDefault`. There are versions for extracting values as `string`, `boolean`, `int`, or a `enum`. This might be helpful in other areas or not.

> REMEMBER: When changing `App.config` locally, make sure you rebuild otherwise your changes won't be picked up.

Extraction

KPI data points are extracted from the source systems via `SourceExtractor`. This class is a wrapper class that

aggregates data points from other systems via classes that implement `IKpiExtractor`. Currently only two other systems are accessed, Jira (`JiraKpiExtractor`) and SonarQube (`SonarQubeKpiExtractor`). Regardless of the root system, all data points are sent to RabbitMQ using the same format.

Dimension Tables

The `IKpiExtractor`s leverage source data stored within dimension tables within `Meta`. As a result, rather than the ETL process managing those tables, the data is updated via migration scripts. There are two dimension tables that the KPIs are concerned about.

To assist in reading this source data, the `DimRepository` was created. This provides a very simple ORM for reading dimension tables. This may or may not be useful in other areas of Meta.

Dim_KPI_Components

This table stores information about the UX components that we are gathering data on. This table allows us to link various keys across different source systems. This table contains the following columns:

Name	Description
---	---
Id __ (PK) __	Internal identifier for the Component.
Name	User friendly name for the Component.
JiraModule	The value within the Module Group within Jira that represents this Component.
Repository	The name of the Azure DevOps repository that contains the code for this Component. Currently this is only used to pull data from SonarQube.

Dim_KPI_Metrics

This table stores information about the individual metrics that are captured for each Component. This table allows us to

add certain metrics automatically (SonarQube specifically). This table contains the following columns:

|Name|Description|

|---|---|

|Id __ (PK) __|Internal identifier of the Metric.|

|Key|The key used to identify this Metric. This may correspond to keys provided by external systems, or it may be used internally to calculate a metric.|

|Source|Indicates where this metric comes from. (Jira = 0; SonarQube = 1)|

IKpiExtractor.cs (Monitoring.ETL.Domain\Kpis\Source\IKpiExtractor.cs):

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Domain.Kpis.Source.Models;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.Kpis.Source

{

public interface IKpiExtractor

{

string Name { get; }

IDimRepository DimRepository { get; set; }

```
Task<ResultSet<KpiDataPoint>> ExtractDataPoints(CancellationToken cancellationToken, IEtlProcessLogger
logger);

    }

}
```

ISourceDelayFactory.cs (Monitoring.ETL.Domain\Kpis\Source\ISourceDelayFactory.cs):

```
using System;
```

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Kpis.Source
```

```
{

    public interface ISourceDelayFactory

    {

        TimeSpan DetermineOneDayDelay(DateTime dateTime);

    }

}
```

JiraKpiExtractor.cs (Monitoring.ETL.Domain\Kpis\Source\Jira\JiraKpiExtractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Configuration;
```

```
using System.IO;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using Atlassian.Jira;
```

```
using ETL.Process;

using Microsoft.SqlServer.Server;

using Monitoring.ETL.Domain.Kpis.Source.Models;

using Monitoring.ETL.Domain.Kpis.Source;

using Monitoring.ETL.Process;

using Newtonsoft.Json.Linq;


namespace Monitoring.ETL.Domain.Kpis.Jira

{

    /// <summary>

    /// Extract KPI data from Jira using the Jira client and returns the result set

    /// </summary>

    public class JiraKpiExtractor : IKpiExtractor

    {

        private const int daysPerSprint = 14;

        private const int numberOfSprints = 2;

        public const string CONFIG_JIRA_URL = "Jira:Url";

        public const string DEFAULT_JIRA_URL = "https://plexsystems.atlassian.net";

        public const string CONFIG_JIRA_USERNAME = "Jira:Username";

        public const string CONFIG_JIRA_TOKEN = "Jira:Token";


        public const string PBIS_CLOSED_LAST_SPRINTS = "pbis_closed_last_sprints";

        public const string PBIS_CLOSED_LAST_SPRINTS_ONTIME = "pbis_closed_last_sprints_ontime";

        public const string PBIS_CLOSED_LAST_SPRINTS_OVERDUE = "pbis_closed_last_sprints_overdue";

        public const string TECH_DEBT_CLOSED_LAST_SPRINTS = "tech_debt_closed_last_sprints";

        public const string TECH_DEBT_OPENED_LAST_SPRINTS = "tech_debt_opened_last_sprints";

    }

}
```

```

public const string ROLLBACKS_LAST_SPRINTS = "rollbacks_last_sprints";

public const string PBIS_CLOSED_LAST_SPRINTS_AVERAGE_DURATION =

"pbis_closed_last_sprints_average_duration";


public const string STATUS_DONE = "Done";

public const string SPRINT_FIELD_IDENTIFIER = "customfield_10020";

public const string PRODUCT_SOURCE_IDENTIFER = "customfield_10035"; //I think this is the product source

public const int PRODUCT_SOURCE_TECH_DEBT_IDENTIFER = 40; //I think this is the product source


public static readonly IReadOnlyDictionary<string, Func<Issue, bool>> MetricPredicates =

    new Dictionary<string, Func<Issue, bool>>()

{

    { PBIS_CLOSED_LAST_SPRINTS, issue => issue.Status.Name == STATUS_DONE },

    { PBIS_CLOSED_LAST_SPRINTS_ONTIME, issue => issue.Status.Name == STATUS_DONE &&

issue.AdditionalFields[SPRINT_FIELD_IDENTIFIER].Count() <= numberOfSprints },

    { PBIS_CLOSED_LAST_SPRINTS_OVERDUE, issue => issue.Status.Name == STATUS_DONE &&

issue.AdditionalFields[SPRINT_FIELD_IDENTIFIER].Count() > numberOfSprints },

    { TECH_DEBT_CLOSED_LAST_SPRINTS, issue => issue.Status.Name == STATUS_DONE &&

issue.AdditionalFields.TryGetValue(PRODUCT_SOURCE_IDENTIFER, out JToken token) && token.HasValues &&

token.Value<int>("id") == PRODUCT_SOURCE_TECH_DEBT_IDENTIFER },

    { ROLLBACKS_LAST_SPRINTS, issue => true },

    { TECH_DEBT_OPENED_LAST_SPRINTS, issue => true },

};


private readonly Atlassian.Jira.Jira _jira;

```



```
public string Name
```

```
{
```

```
    get { return "Jira"; }
```

```
}
```

```
public IDimRepository DimRepository { get; set; }
```

```
public JiraKpiExtractor()
```

```
{
```

```
    _jira = Atlassian.Jira.Jira.CreateRestClient(
```

```
        ConfigurationManager.AppSettings.GetValueOrDefault(CONFIG_JIRA_URL, DEFAULT_JIRA_URL),
```

```
        ConfigurationManager.AppSettings[CONFIG_JIRA_USERNAME],
```

```
        ConfigurationManager.AppSettings[CONFIG_JIRA_TOKEN]);
```

```
}
```

```
public async Task<ResultSet<KpiDataPoint>> ExtractDataPoints(Cancellation token cancellationToken,
```

```
IEtlProcessLogger logger)
```

```
{
```

```
    if (DimRepository == null)
```

```
    {
```

```
        throw new ArgumentNullException(nameof(DimRepository), $"Please set the
```

```
{nameof(JiraKpiExtractor.DimRepository)} property before running the extraction.");
```

```
    }
```

```
    var resultSet = new ResultSet<KpiDataPoint>();
```

```

List<Component> components = await DimRepository.LoadAll<Component>();

List<Metric> metrics = await DimRepository.LoadAll<Metric>(m => m.Source == MetricSources.Jira);

foreach (var component in components.Where(c => c.JiraModule != null))
{
    try
    {
        ((List<KpiDataPoint>)resultSet.Results).AddRange(await LoadMetrics(component, metrics));

        logger.Information($"Extracted Jira metrics for {component.Name}");
    }
    catch (Exception ex)
    {
        ex.Data.Add("Component", component.Name);

        resultSet.Exceptions.Add(ex);
    }
}

return resultSet;
}

```

```

private async Task<List<KpiDataPoint>> LoadMetrics(Component component, List<Metric> metrics)
{
    int days = numberOfSprints * daysPerSprint;

    List<KpiDataPoint> datapoints = new List<KpiDataPoint>();

    if (component.JiraModule != null && component.Repository != null)

```

```
{  
  
    datapoints.AddRange(await AddSprintMetrics(metrics, component, days));  
  
    datapoints.Add(await AddTechDebtMetrics(metrics, component, days));  
  
    datapoints.Add(await AddRollbackMetrics(metrics, component, days));  
  
}
```

```
return datapoints;
```

```
}
```

```
private async Task<IPagedQueryResult<Issue>> QueryJira(string jql)
```

```
{
```

```
    IssueSearchOptions so = new IssueSearchOptions(jql)
```

```
{
```

```
    MaxIssuesPerRequest = 100,
```

```
    AdditionalFields = new List<string>()
```

```
{
```

```
    "**all",
```

```
    "**navigable"
```

```
}
```

```
};
```

```
return await _jira.Issues.GetIssuesFromJqlAsync(so);
```

```
}
```

```
private async Task<KpiDataPoint> AddTechDebtMetrics(List<Metric> metrics, Component component, int  
numberOfPreviousDays)
```

```
{
```

```
string jql = $"type in (Story, Bug, Defect) AND (\"Module Group[Dropdown]\" = \"{component.JiraModule}\" OR  
\"Repositories[Labels]\" in ({component.Repository})) AND Account = \"Technical Debt\" AND createdAt >  
startOfDay(-{numberOfPreviousDays})";
```

```
List<Issue> issues = (await (QueryJira(jql))).ToList();
```

```
return LoadCountBasedMetric(issues, metrics, component, TECH_DEBT_OPENED_LAST_SPRINTS);
```

```
}
```

```
private async Task<KpiDataPoint> AddRollbackMetrics(List<Metric> metrics, Component component, int  
numberOfPreviousDays)
```

```
{
```

```
string jql = $"(filter = \"TCIF Customer Issues\") AND type != Defect AND (resolution = rollback OR \"Defect Source\"  
= Regression) AND \"Module Group[Dropdown]\" = \"{component.JiraModule}\" AND createdAt >  
startOfDay(-{numberOfPreviousDays})";
```

```
List<Issue> issues = (await (QueryJira(jql))).ToList();
```

```
return LoadCountBasedMetric(issues, metrics, component, ROLLBACKS_LAST_SPRINTS);
```

```
}
```

```
private async Task<List<KpiDataPoint>> AddSprintMetrics(List<Metric> metrics, Component component, int  
numberOfPreviousDays)
```

```
{
```

```
string jql = $"type in (Story, Bug, Defect) AND resolution In (Done) AND (\"Module Group[Dropdown]\" =  
 \"{component.JiraModule}\" OR \"Repositories[Labels]\" in ({component.Repository})) AND resolutiondate >  
startOfDay(-{numberOfPreviousDays})";
```

```
List<Issue> issues = (await (QueryJira(jql))).ToList();
```

```

var datapoints = new List<KpiDataPoint>

{
    LoadCountBasedMetric(issues, metrics, component, PBIS_CLOSED_LAST_SPRINTS),
    LoadCountBasedMetric(issues, metrics, component, PBIS_CLOSED_LAST_SPRINTS_ONTIME),
    LoadCountBasedMetric(issues, metrics, component, PBIS_CLOSED_LAST_SPRINTS_OVERDUE),
    LoadCountBasedMetric(issues, metrics, component, TECH_DEBT_CLOSED_LAST_SPRINTS)
};

int id = GetMetricIdFromKey(metrics, PBIS_CLOSED_LAST_SPRINTS_AVERAGE_DURATION);

datapoints.Add(new KpiDataPoint
{
    ComponentId = component.Id,
    MetricId = id,
    Value = !issues.Any() ?
        (double?)null :
        issues.Where(i => i.Status.Name == STATUS_DONE).Average(i =>
i.AdditionalFields["customfield_10020"].Count()),
    Timestamp = DateTime.Today
});

return datapoints;
}

private KpiDataPoint LoadCountBasedMetric(List<Issue> issues, List<Metric> metrics, Component component, string
metricName)
{

```

```

return new KpiDataPoint
{
    ComponentId = component.Id,
    MetricId = GetMetricIdFromKey(metrics, metricName),
    Value = issues.Count(MetricPredicates[metricName]),
    Timestamp = DateTime.Today
};
}

private int GetMetricIdFromKey(List<Metric> metrics, string key)
{
    int? id = metrics.FirstOrDefault(m => m.Key == key)?.Id;
    if (!id.HasValue)
    {
        throw new InvalidDataException($"Metric {key} not in the database.");
    }

    return id.Value;
}

}

}

```

Component.cs (Monitoring.ETL.Domain\Kpis\Source\Models\Component.cs):

```
using System.ComponentModel.DataAnnotations.Schema;
```

```
namespace Monitoring.ETL.Domain.Kpis.Source.Models
```

```
{
```

```
[Table("Dim_KPI_Components")]
```

```
public class Component
```

```
{
```

```
public int Id { get; set; }
```

```
public string Name { get; set; }
```

```
public string JiraModule { get; set; } = null;
```

```
public string Repository { get; set; } = null;
```

```
}
```

```
}
```

KpiDataPoint.cs (Monitoring.ETL.Domain\Kpis\Source\Models\KpiDataPoint.cs):

```
using System;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.RabbitMQ;
```

```
using Newtonsoft.Json;
```

```
using Newtonsoft.Json.Serialization;
```

```
namespace Monitoring.ETL.Domain.Kpis.Source.Models
```

```
{
```

```
public class KpiDataPoint : ILoadModel, IExtractModel
```

```
{
```

```
[JsonIgnore]
```

```
public string JsonMessage
```

```
{
```

```
get

{

    return GenerateJson();

}

set

{

    ParseJson(value);

}

}
```

```
public int ComponentId { get; set; }
```

```
public int MetricId { get; set; }
```

```
public double? Value { get; set; }
```

```
public DateTime Timestamp { get; set; }
```

```
private void ParseJson(string json)

{

    KpiDataPoint kdp = JsonConvert.DeserializeObject<KpiDataPoint>(json);

    ComponentId = kdp.ComponentId;

    MetricId = kdp.MetricId;

    Value = kdp.Value;

    Timestamp = kdp.Timestamp;

}
```



```

private string GenerateJson()
{
    return JsonConvert.SerializeObject(
        this,
        new JsonSerializerSettings
        {
            NullValueHandling = NullValueHandling.Include,
            Formatting = Formatting.Indented,
            ContractResolver = new CamelCasePropertyNamesContractResolver(),
            DateFormatString = "O",
            FloatParseHandling = FloatParseHandling.Double
        });
}
}
}

```

Metric.cs (Monitoring.ETL.Domain\Kpis\Source\Models\Metric.cs):

```
using System.ComponentModel.DataAnnotations.Schema;
```

```
namespace Monitoring.ETL.Domain.Kpis.Source.Models
```

```

{
    [Table("Dim_KPI_Metrics")]

    public class Metric
    {
        public int Id { get; set; }
    }
}

```

```
        public string Key { get; set; }

        public MetricSources Source { get; set; }

    }

}
```

MetricSources.cs (Monitoring.ETL.Domain\Kpis\Source\Models\MetricSources.cs):

```
namespace Monitoring.ETL.Domain.Kpis.Source.Models

{

    public enum MetricSources

    {

        Jira = 0,

        SonarQube = 1

    }

}
```

SonarQubeKpiExtractor.cs (Monitoring.ETL.Domain\Kpis\Source\SonarQube\SonarQubeKpiExtractor.cs):

```
using System;

using System.Collections.Generic;

using System.Configuration;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

using SonarQube.Net;

using SonarQubeAuth = SonarQube.Net.Common.Authentication;
```

```
using SonarQube.Net.Models;

using Monitoring.ETL.Domain.Kpis.Jira;

using Monitoring.ETL.Domain.Kpis.Source.Models;

using Component = Monitoring.ETL.Domain.Kpis.Source.Models.Component;

using Metric = Monitoring.ETL.Domain.Kpis.Source.Models.Metric;


namespace Monitoring.ETL.Domain.Kpis.Source.SonarQube

{

    public class SonarQubeKpiExtractor : IKpiExtractor

    {

        public const string DEFAULT_SONARQUBE_URL = "https://sonarqube.aze.plex";

        public const string CONFIG_SONARQUBE_URL = "SonarQube:Url";

        public const string CONFIG_SONARQUBE_TOKEN = "SonarQube:Token";


        private readonly SonarQubeClient _client;


        public string Name

        {

            get { return "SonarQube"; }

        }


        public IDimRepository DimRepository { get; set; }


        public SonarQubeKpiExtractor()

        {

            _client = new SonarQubeClient(
```

```

        ConfigurationManager.AppSettings.GetValueOrDefault(CONFIG_SONARQUBE_URL,
DEFAULT_SONARQUBE_URL),

        new SonarQubeAuth.BasicAuthentication(

            ConfigurationManager.AppSettings[CONFIG_SONARQUBE_TOKEN],

            string.Empty));
    }

    public async Task<ResultSet<KpiDataPoint>> ExtractDataPoints(CancellationTok
en cancellationToken,
IEtlProcessLogger logger)

    {

        if (DimRepository == null)

        {

            throw new ArgumentNullException(nameof(DimRepository), $"Please set the
{nameof(JiraKpiExtractor.DimRepository)} property before running the extraction.");

        }

        var resultSet = new ResultSet<KpiDataPoint>();

        List<Component> components = await DimRepository.LoadAll<Component>();

        List<Metric> metrics = await DimRepository.LoadAll<Metric>(m => m.Source == MetricSources.SonarQube);

        foreach (var component in components.Where(c => c.Repository != null))

        {

            try

            {

                ((List<KpiDataPoint>)resultSet.Results).AddRange(await LoadMetrics(component, metrics));

                logger.Information($"Extracted SonarQube metrics for {component.Name}");

            }

        }

```

```

        catch (Exception ex)

        {

            ex.Data.Add("Component", component.Name);

            resultSet.Exceptions.Add(ex);

        }

    }

    return resultSet;

}

private async Task<List<KpiDataPoint>> LoadMetrics(Component component, List<Metric> metrics)

{

    List<KpiDataPoint> datapoints = new List<KpiDataPoint>();

    ComponentMeasure measures = await _client.GetMeasuresComponentAsync(component.Repository,

metrics.Select(m => m.Key).ToArray());

    foreach (var metric in metrics)

    {

        string value = measures.Component.Measures.FirstOrDefault(m => m.Metric == metric.Key)?.Value;

        datapoints.Add(new KpiDataPoint

        {

            ComponentId = component.Id,

            MetricId = metric.Id,

            Value = double.TryParse(value, out double parsedValue) ? parsedValue : (double?)null,

            Timestamp = DateTime.Today
        });
    }
}

```

```

    });

}

return datapoints;

}

}

}

```

SourceDelayFactory.cs (Monitoring.ETL.Domain\Kpis\Source\SourceDelayFactory.cs):

```

using System;

using System.Configuration;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Domain.ElasticSearch;

using Monitoring.ETL.Domain.Kpis.Source.Models;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.Kpis.Source
{
    public class SourceDelayFactory : IExtractionDelayFactory<KpiDataPoint>, ISourceDelayFactory
    {
        public const string CONFIG_RUN_NOW = "Kpis:RunNow";

        public const string CONFIG_NEXT_RUN = "Kpis:NextRun";

        public const string CONFIG_BUILD_DAY = "Kpis:BuildDay";

        public const string CONFIG_BUILD_INTERVAL_WEEKS = "Kpis:BuidIntervalWeeks";
    }
}

```

```

public const string CONFIG_KPI_RUN_HOUR = "Kpis:RunHour";

public const int DEFAULT_BUILD_INTERVAL_WEEKS = 2;

public const int DEFAULT_KPI_RUN_HOUR = 12;

public const DayOfWeek DEFAULT_BUILD_DAY = DayOfWeek.Tuesday;


public async Task Create(ResultSet<KpiDataPoint> results, CancellationToken cancellationToken, IEtlProcessLogger
logger)

{

    var timeSpan = DetermineOneDayDelay(DateTime.Today);

    logger.Information($"Sleeping for {timeSpan}");

    await Task.Delay(timeSpan, cancellationToken);

}


public TimeSpan DetermineOneDayDelay(DateTime dateTime)

{

    int hourToRunKpis = ConfigurationManager.AppSettings.GetValueOrDefault(CONFIG_KPI_RUN_HOUR,
DEFAULT_KPI_RUN_HOUR);

    hourToRunKpis = hourToRunKpis < 24 ? hourToRunKpis : DEFAULT_KPI_RUN_HOUR;

    DateTime nextRun = new DateTime(dateTime.Year,

        dateTime.Month,

        dateTime.Day,

        hourToRunKpis,

        0,

        0,

        dateTime.Kind);

```

```
        if (dateTime.Hour >= hourToRunKpis)
        {
            nextRun = nextRun.AddDays(1);
        }

        return nextRun.Subtract(dateTime);
    }
}
}
```

SourceExtractor.cs (Monitoring.ETL.Domain\Kpis\Source\SourceExtractor.cs):

```
using System;

using System.Collections.Generic;

using System.Configuration;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Domain.ElasticSearch;

using Monitoring.ETL.Domain.Kpis.Source.Models;

using Monitoring.ETL.Domain.Warehouse;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.Kpis.Source
{
    /// <summary>
```



```
/// Extract KPI data
```

```
/// </summary>
```

```
public class SourceExtractor : IExtractor<KpiDataPoint>
```

```
{
```

```
    private readonly List<IKpiExtractor> _kpiExtractors;
```

```
    private readonly IDimRepository _dimRepository;
```

```
    private readonly ILoadStateRepository _loadStateRepository;
```

```
/// <summary>
```

```
/// Normal constructor for running code
```

```
/// </summary>
```

```
public SourceExtractor()
```

```
    : this(
```

```
        new List<IKpiExtractor>()
```

```
    {
```

```
        new Jira.JiraKpiExtractor(),
```

```
        new SonarQube.SonarQubeKpiExtractor()
```

```
    },
```

```
    new DimRepository(new LoadStateRepository(new WarehouseServerProvider()).GetConnection()),
```

```
    new LoadStateRepository<KpiDataPoint>())
```

```
{
```

```
}
```

```
/// <summary>
```

```
/// Additional constructor for testing
```

```
/// </summary>
```

```

public SourceExtractor(List<IKpiExtractor> kpiExtractors, IDimRepository dimRepository, ILoadStateRepository
loadStateRepository)

{
    _kpiExtractors = kpiExtractors;

    _dimRepository = dimRepository;

    _loadStateRepository = loadStateRepository;
}

public async Task<ResultSet<KpiDataPoint>> ExtractAllSinceLastExtractAsync(CancellationTok
cancellationToken, IEtlProcessLogger logger)

{
    var resultSet = new ResultSet<KpiDataPoint>();

    var lastRunDate = _loadStateRepository.GetDate();

    int weeks =
ConfigurationManager.AppSettings.GetValueOrDefault(SourceDelayFactory.CONFIG_BUILD_INTERVAL_WEEKS,
SourceDelayFactory.DEFAULT_BUILD_INTERVAL_WEEKS);

    var expectedNextRunDate = lastRunDate.AddDays(weeks * 7);

    if (expectedNextRunDate.Date > DateTime.Now.Date)

    {
        logger.Information($"Not scheduled to run until {expectedNextRunDate.Date}.");

        return resultSet;
    }

    foreach (var extractor in _kpiExtractors)

```

```

{

    extractor.DimRepository = _dimRepository;

    try

    {

        var extractorResults = await extractor.ExtractDataPoints(cancellationToken, logger);

        ((List<KpiDataPoint>)resultSet.Results).AddRange(extractorResults.Results);

        ((List<Exception>)resultSet.Exceptions).AddRange(extractorResults.Exceptions);

        logger.Information($"Extracted KPI metrics for {extractor.Name}");

    }

    catch (Exception ex)

    {

        ex.Data.Add("Extractor", extractor.Name);

        resultSet.Exceptions.Add(ex);

    }

}

await _loadStateRepository.SetDate(DateTime.Now.Date);

return resultSet;

}

public Task<IEnumerable<KpiDataPoint>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
CancellationTokens cancellationTokens, IEtlProcessLogger logger)

{

    throw new NotImplementedException();

}

```

```
}
```

```
}
```

SourceRabbitLoader.cs (Monitoring.ETL.Domain\Kpis\Source\SourceRabbitLoader.cs):

```
using Monitoring.ETL.Domain.Kpis.Source.Models;
```

```
using Monitoring.ETL.Domain.RabbitMQ.Model;
```

```
namespace Monitoring.ETL.Domain.Kpis.Source
```

```
{  
  
    public class SourceRabbitLoader : RabbitMQ.Loader<RabbitMQKpiDataPoint>  
  
    {  
  
        public SourceRabbitLoader()  
  
            : base(new RabbitMQ.Kpis.Producer())  
  
        {  
  
        }  
  
    }  
  
}
```

SourceTransformer.cs (Monitoring.ETL.Domain\Kpis\Source\SourceTransformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.Kpis.Source.Models;
```

```
using Monitoring.ETL.Domain.RabbitMQ.Model;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Kpis.Source
```

```
{
```

```
    public class SourceTransformer : ITransformer<KpiDataPoint, RabbitMQKpiDataPoint>
```

```
    {
```

```
        public async Task<IEnumerable<RabbitMQKpiDataPoint>> TransformAsync(IEnumerable<KpiDataPoint> extracted,  
CancellationTokens cancellationTokens, IEtlProcessLogger logger)
```

```
    {
```

```
        await Task.Yield();
```

```
        return extracted.Select(pc => new RabbitMQKpiDataPoint
```

```
        {
```

```
            JsonMessage = pc.JsonMessage
```

```
        });
```

```
    }
```

```
}
```

```
}
```

KpiRabbitExtractor.cs (Monitoring.ETL.Domain\Kpis\Warehouse\KpiRabbitExtractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.RabbitMQ.Model;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Kpis.Warehouse
```

```
{
```

```
    public class KpiRabbitExtractor : IExtractor<RabbitMQKpiDataPoint>
```

```
    {
```

```
        private RabbitMQ.Kpis.Consumer _consumer;
```

```
        public async Task<ResultSet<RabbitMQKpiDataPoint>> ExtractAllSinceLastExtractAsync(CancellationTokencancellationTokentoken, IEtlProcessLogger logger)
```

```
        {
```

```
            _consumer = _consumer ?? new RabbitMQ.Kpis.Consumer(logger);
```

```
            return await _consumer.ExtractAsync();
```

```
        }
```

```
        public Task<IEnumerable<RabbitMQKpiDataPoint>> ExtractBetweenAsync(DateTime startDate, DateTimeendDate, CancellationTokentoken, IEtlProcessLogger logger)
```

```
        {
```

```
            throw new NotImplementedException();
```

```
        }
```

```
    }
```

```
}
```

KpiWarehouseLoader.cs (Monitoring.ETL.Domain\Kpis\Warehouse\KpiWarehouseLoader.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```
using WarehouseModels = Monitoring.ETL.Domain.Warehouse.Model;
```

```
namespace Monitoring.ETL.Domain.Kpis.Warehouse
```

```
{  
  
    public class KpiWarehouseLoader : Loader<WarehouseModels.Fact_KPI_DataPoints_w>  
  
    {  
  
        public KpiWarehouseLoader() : base("Fact_KPI_DataPoints_Add")  
  
        {  
  
  
  
  
        }  
  
    }  
}
```

WarehouseDelayFactory.cs (Monitoring.ETL.Domain\Kpis\Warehouse\WarehouseDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Kpis.Warehouse
```

```
{  
  
    public class WarehouseDelayFactory : PeriodicExtractionDelayFactory<RabbitMQ.Model.RabbitMQKpiDataPoint>  
  
    {  
  
        protected override TimeSpan DelayTimeSpan => new TimeSpan(0, 0, 10);  
  
    }  
}
```

WarehouseTransformer.cs (Monitoring.ETL.Domain\Kpis\Warehouse\WarehouseTransformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

using WarehouseModels = Monitoring.ETL.Domain.Warehouse.Model;

namespace Monitoring.ETL.Domain.Kpis.Warehouse
{
    public class WarehouseTransformer : ITransformer<RabbitMQ.Model.RabbitMQKpiDataPoint,
WarehouseModels.Fact_KPI_DataPoints_w>
    {
        public async Task<IEnumerable<WarehouseModels.Fact_KPI_DataPoints_w>>
TransformAsync(IEnumerable<RabbitMQ.Model.RabbitMQKpiDataPoint> extracted, CancellationToken
cancellationToken, IEtlProcessLogger logger)
        {
            await Task.Yield();

            return extracted.Select(pc => new WarehouseModels.Fact_KPI_DataPoints_w
            {
                JSON_Message = pc.JsonMessage
            });
        }
    }
}

```

LoginExtractModel.cs (Monitoring.ETL.Domain\Login\Extract>LoginExtractModel.cs):


```
using System;
```

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Login.Extract
```

```
{
```

```
    public class LoginExtractModel : IExtractModel
```

```
    {
```

```
        public string IpAddress { get; set; }
```

```
        public DateTime LoginDate { get; set; }
```

```
        public long LoginNo { get; set; }
```

```
        public string LoginOrigin { get; set; }
```

```
        public int PUN { get; set; }
```

```
        [Obsolete]
```

```
        public string SqlServer { get; set; }
```

```
        public string SqlServerInstanceName { get; set; }
```

```
        public string SqlServerMachineName { get; set; }
```

```
        public int SqlServerPort { get; set; }
```

```
    }
```

```
}
```

LoginNscaleExtractor.cs (Monitoring.ETL.Domain\Login\Extract\LoginNscaleExtractor.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.ElasticSearch;
```

```
using Monitoring.ETL.Process;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Data;
```

```
using System.Data.SqlClient;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.Login.Extract
```

```
{
```

```
    public sealed class LoginNscaleExtractor : IExtractor<LoginExtractModel>
```

```
    {
```

```
        private const string TrustedSqlServerConnectionStringTemplate = "data source={0};initial
```

```
catalog=Plexus_Control;integrated security=True;";
```

```
        private readonly ServiceToolConnectionRepository _serviceToolConnectionRepository = new
```

```
ServiceToolConnectionRepository();
```

```
        private readonly LoadStateRepository _state = new LoadStateRepository();
```

```
        public async Task<ResultSet<LoginExtractModel>> ExtractAllSinceLastExtractAsync(CancellationTok
```

```
cancellationTok
```

```
en, IETLProcessLogger logger)
```

```
    {  
        var results = new ResultSet<LoginExtractModel>();
```

```

var loadStateUpdates = new Dictionary<string, DateTime>();

try
{
    var servers = _serviceToolConnectionRepository.GetServersToProcess("Login ETL").OrderBy(kvp =>
kvp.Key).ToList();

    foreach (var server in servers)
    {
        var serviceName = server.Key;

        var sqlServerModel = server.Value;

        logger.Information($"Processing SQL Server Instance: {serviceName} ({sqlServerModel.MachineName})
...");

        string loadName = $"Login:{serviceName}";

        var lastDate = _state.GetDateFor(loadName);

        if (lastDate.Year < 2000)
        {
            lastDate = new DateTime(2017, 1, 1);
        }

        try
        {
            var serverResults = GetLoginsForServer(sqlServerModel, serviceName, lastDate);

```

```

logger.Information($"Success - {serverResults?.Count} record(s) found.");

if (serverResults?.Count > 0)
{
    loadStateUpdates.Add(loadName, serverResults.Select(l => l.LoginDate).Max());

    foreach (var loginExtractModel in serverResults)
    {
        results.Results.Add(loginExtractModel);
    }
}

catch (Exception e)
{
    if (e.IsNetworkConnectionException())
    {
        logger.Information("Instance currently unavailable. Skipping extraction for this execution.");
    }
    else
    {
        logger.Information("Failed!");

        e.Data.Add("X-Server", server);

        results.Exceptions.Add(e);
    }
}

```

```
// If cancellation is requested, abort and return an empty result set.
```

```
// Note that we are NOT updating the load states otherwise we would have data loss.
```

```
if (cancellationToken.IsCancellationRequested)
```

```
{
```

```
    logger.Information("Cancellation requested - discarding results and exiting.");
```

```
    return new ResultSet<LoginExtractModel>();
```

```
}
```

```
}
```

```
// After all servers have been processed, persist the keys to the load states.
```

```
foreach (var loadStateUpdate in loadStateUpdates)
```

```
{
```

```
    await _state.SetStateAsync(loadStateUpdate.Key, loadStateUpdate.Value);
```

```
}
```

```
return results;
```

```
}
```

```
catch (Exception e)
```

```
{
```

```
    results.Exceptions.Add(e);
```

```
    return results;
```

```
}
```

```
}
```

```
public Task<IEnumerable<LoginExtractModel>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
```

```
CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    throw new NotImplementedException();  
  
}
```

```
private static List<LoginExtractModel> GetLoginsForServer(ServiceToolConnectionRepository.SqlServerModel  
sqlServerModel, string serviceName, DateTime lastLoginDate)
```

```
{  
  
    var logins = new List<LoginExtractModel>();  
  
    var serverName = sqlServerModel.MachineName + (sqlServerModel.Port > 0 ? $",{sqlServerModel.Port}" :  
$"\{sqlServerModel.InstanceName}");  
  
    using (var con = new SqlConnection(string.Format(TrustedSqlServerConnectionStringTemplate, serverName)))  
    {  
  
        con.Open();  
  
        using (var cmd = new SqlCommand("Plexus_Control.dbo.Logins_Extract_Get", con))  
        {  
  
            cmd.CommandType = CommandType.StoredProcedure;  
  
            cmd.Parameters.Add(  
  
                new SqlParameter  
  
                {  
  
                    ParameterName = "@After_Date",  
  
  
  
                    // Move to minute after the last collected minute.  
  
                    Value = lastLoginDate.AddMinutes(1),  

```

```

        DbType = DbType.DateTime

    });

using (var reader = cmd.ExecuteReader())

{

    if (reader.HasRows)

    {

        var loginNoOrdinal = reader.GetOrdinal("Login_No");

        var punOrdinal = reader.GetOrdinal("Plexus_User_No");

        var loginDateOrdinal = reader.GetOrdinal("Login_Date");

        var ipAddressOrdinal = reader.GetOrdinal("IP_Address");

        var loginOriginOrdinal = reader.GetOrdinal("Login_Origin");

        while (reader.Read())

        {

            var loginModel = new LoginExtractModel

            {

                IpAddress = reader.IsDBNull(ipAddressOrdinal) ? null : reader.GetString(ipAddressOrdinal),

                LoginDate = reader.GetDateTime(loginDateOrdinal),

                LoginNo = Convert.ToInt64(reader.GetValue(loginNoOrdinal)),

                LoginOrigin = reader.IsDBNull(loginOriginOrdinal) ? null : reader.GetString(loginOriginOrdinal),

                PUN = reader.GetInt32(punOrdinal),

                // Leaving in place until the new columns are fully deployed.

                SqlServer = serviceName,
            }
        }
    }
}

```

```

        SqlServerMachineName = sqlServerModel.MachineName,

        SqlServerInstanceName = sqlServerModel.InstanceName,

        SqlServerPort = sqlServerModel.Port

    };

    logins.Add(loginModel);

    }

    }

    }

    }

    }

    return logins;

}

}

}

```

LoginRabbitMQExtractor.cs (Monitoring.ETL.Domain>Login\Extract>LoginRabbitMQExtractor.cs):

```

using ETL.Process;

using Monitoring.ETL.Domain.Login.Load;

using Monitoring.ETL.Domain.RabbitMQ.Login;

using Monitoring.ETL.Process;

using System;

using System.Collections.Generic;

```



```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.Login.Extract
```

```
{
```

```
    public sealed class LoginRabbitMQExtractor : IExtractor<FactLoginWarehouseLoadModel>
```

```
    {
```

```
        private LoginConsumer _loginConsumer;
```

```
        public async Task<ResultSet<FactLoginWarehouseLoadModel>>
```

```
ExtractAllSinceLastExtractAsync(CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
    {
```

```
        _loginConsumer = _loginConsumer ?? new LoginConsumer(logger);
```

```
        return await _loginConsumer.ExtractAsync();
```

```
    }
```

```
        public Task<IEnumerable<FactLoginWarehouseLoadModel>> ExtractBetweenAsync(DateTime startDate,
```

```
DateTime endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
    {
```

```
        _loginConsumer = _loginConsumer ?? new LoginConsumer(logger);
```

```
        throw new NotImplementedException();
```

```
    }
```

```
}
```

```
}
```

```
FactLoginWarehouseExtractionDelayFactory.cs
```

(Monitoring.ETL.Domain\Login\FactLoginWarehouseExtractionDelayFactory.cs):

using System;

using Monitoring.ETL.Domain.Login.Load;

namespace Monitoring.ETL.Domain.Login

```
{  
  
    public sealed class FactLoginWarehouseExtractionDelayFactory :  
ThresholdExtractionDelayFactory<FactLoginWarehouseLoadModel>  
  
    {  
  
        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(2);  
  
        protected override int MaxThresholdForDelay => 100;  
  
    }  
}
```

FactLoginLoadModel.cs (Monitoring.ETL.Domain\Login\Load\FactLoginLoadModel.cs):

using ETL.Process;

using System;

namespace Monitoring.ETL.Domain.Login.Load

```
{  
  
    public class FactLoginLoadModel : ILoadModel  
  
    {  
  
        public string IP_Address { get; set; }  
  
        public DateTime Login_Date { get; set; }  
  
    }  
}
```

```
public long? Login_No { get; set; }
```

```
public string Login_Origin { get; set; }
```

```
public int PUN { get; set; }
```

```
[Obsolete("The three \'Source SQL Server\' fields should be used instead of this one.")]
```

```
public string Source_Server { get; set; }
```

```
public string Source_SQL_Server_Instance_Name { get; set; }
```

```
public string Source_SQL_Server_Machine_Name { get; set; }
```

```
public int Source_SQL_Server_Port { get; set; }
```

```
}
```

```
}
```

FactLoginWarehouseLoadModel.cs (Monitoring.ETL.Domain\Login\Load\FactLoginWarehouseLoadModel.cs):

```
using Monitoring.ETL.Domain.RabbitMQ;
```

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.Login.Load
```

```
{
```

```
    public class FactLoginWarehouseLoadModel : FactLoginLoadModel, IRabbitMQExtractModel, IWarehouseLoadModel
```

```
    {
```

```
        public int Warehouse_Load_Key { get; set; }
```

```
    }
```

```
}
```

LoginRabbitMQLoader.cs (Monitoring.ETL.Domain\Login\Load>LoginRabbitMQLoader.cs):

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.RabbitMQ.Login;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Login.Load
```

```
{
```

```
    public class LoginRabbitMQLoader : ILoader<FactLoginLoadModel>
```

```
    {
```

```
        private readonly LoginProducer _loginProducer = new LoginProducer();
```

```
        public async Task<bool> LoadAsync(IEnumerable<FactLoginLoadModel> transformed, CancellationToken
```

```
cancellation_token, IEtlProcessLogger logger)
```

```
    {
```

```
        await Task.Run(
```

```
            () =>
```

```
            {
```

```
                foreach (var factLoginLoadModel in transformed)
```

```
                {
```

```
                    _loginProducer.Publish(factLoginLoadModel);
```

```
                }
```

```
            });
```

```
        return true;

    }

}

}
```

LoginWarehouseLoader.cs (Monitoring.ETL.Domain\Login\Load>LoginWarehouseLoader.cs):

```
using Monitoring.ETL.Domain.Warehouse;

using Monitoring.ETL.Domain.Warehouse.Model;

namespace Monitoring.ETL.Domain.Login.Load
{
    public class LoginWarehouseLoader : Loader<Login_w>
    {
        public LoginWarehouseLoader() : base("Fact_Login_Add", true)
        {
        }
    }
}
```

LoginExtractionDelayFactory.cs (Monitoring.ETL.Domain\Login>LoginExtractionDelayFactory.cs):

```
using Monitoring.ETL.Domain.Login.Extract;

using System;

namespace Monitoring.ETL.Domain.Login
```

```

{

    public sealed class LoginExtractionDelayFactory : ThresholdExtractionDelayFactory<LoginExtractModel>

    {

        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(30);

        protected override int MaxThresholdForDelay => 50;

    }

}

```

LoginRabbitMQTransformer.cs (Monitoring.ETL.Domain\Login\Transform\LoginRabbitMQTransformer.cs):

```

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;


using ETL.Process;


using Monitoring.ETL.Domain.Login.Extract;

using Monitoring.ETL.Domain.Login.Load;

using Monitoring.ETL.Domain.Warehouse.Model;

using Monitoring.ETL.Process;


namespace Monitoring.ETL.Domain.Login.Transform

{

    /// <summary>

    /// Load the messages from RabbitMQ into the table Template_Worktable.

    /// </summary>

```

```

public class LoginRabbitMQTransformer : ITransformer<FactLoginWarehouseLoadModel, Login_w>
{
    public async Task<IEnumerable<Login_w>> TransformAsync(IEnumerable<FactLoginWarehouseLoadModel>
extracted, CancellationToken cancellationToken, IEtlProcessLogger logger)
    {
        await Task.Yield();

        return extracted.Select(pc => new Login_w
        {
            PUN = pc.PUN,
            IP_Address = pc.IP_Address,
            Login_Date = pc.Login_Date,
            Login_No = pc.Login_No,
            Login_Origin = pc.Login_Origin,
            Source_Server = pc.Source_Server,
            Source_SQL_Server_Machine_Name = pc.Source_SQL_Server_Machine_Name,
            Source_SQL_Server_Instance_Name = pc.Source_SQL_Server_Instance_Name,
            Source_SQL_Server_Port = pc.Source_SQL_Server_Port
        });
    }
}

```

LoginTransformer.cs (Monitoring.ETL.Domain\Login\Transform\LoginTransformer.cs):

```

using System.Collections.Generic;

```

```

using System.Linq;

```

```

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Domain.Login.Extract;

using Monitoring.ETL.Domain.Login.Load;

using Monitoring.ETL.Process;


namespace Monitoring.ETL.Domain.Login.Transform
{
    public sealed class LoginTransformer : ITransformer<LoginExtractModel, FactLoginLoadModel>
    {
        public async Task<IEnumerable<FactLoginLoadModel>> TransformAsync(IEnumerable<LoginExtractModel>
extracted, CancellationToken cancellationToken, IEtlProcessLogger logger)
        {
            return await Task.Run(
                () => extracted.Select(
                    I => new FactLoginLoadModel
                    {
                        IP_Address = I.IpAddress,

                        Login_Date = I.LoginDate,

                        Login_Origin = I.LoginOrigin,

                        PUN = I.PUN,

                        Login_No = I.LoginNo,


                        // Leaving in place until the new fields are fully rolled out.

                        Source_Server = I.SqlServer,

```



```

        Source_SQL_Server_Machine_Name = I.SqlServerMachineName,

        Source_SQL_Server_Instance_Name = I.SqlServerInstanceName,

        Source_SQL_Server_Port = I.SqlServerPort

    }},

    cancellationToken);

}

}

}

```

MissingDllHack.cs (Monitoring.ETL.Domain\MissingDllHack.cs):

```

using System.Data.Entity.SqlServer;

namespace Monitoring.ETL.Domain
{
    /// <summary>

    /// Provides a hack which references EntityFramework.SqlServer.dll so that this dll will be

    /// included in the output folder of referencing projects without requiring a direct dependency

    /// on Entity Framework. See http://stackoverflow.com/a/22315164/1141360.

    /// </summary>

    internal static class MissingDllHack
    {
        //

        private static SqlProviderServices _instance = SqlProviderServices.Instance;

    }

}

```

Application_Event_w.cs (Monitoring.ETL.Domain\Model\Application_Event_w.cs):

////-----

/// <auto-generated>

/// This code was generated from a template.

///

/// Manual changes to this file may cause unexpected behavior in your application.

/// Manual changes to this file will be overwritten if the code is regenerated.

/// </auto-generated>

////-----

//namespace Monitoring.ETL.Domain.Model

//{

// using System;

// using System.Collections.Generic;

// public partial class Application_Event_w

// {

// public int Warehouse_Load_Key { get; set; }

// public int Load_Event_Number { get; set; }

// public Nullable<int> Error_Key { get; set; }

// public Nullable<int> PUN { get; set; }

// public Nullable<int> PCN { get; set; }

// public System.DateTime Event_Date { get; set; }

// public string Message { get; set; }

// public string Event_Type { get; set; }

// public string Event_Severity { get; set; }

```
// public string Data_Center { get; set; }

// public string Environment { get; set; }

// public string Service_Application_Name { get; set; }

// public string Filename { get; set; }

// public string Path_Info { get; set; }

// public string HTTP_Referer { get; set; }

// public string Referer_Path { get; set; }

// public string Web_Server { get; set; }

// public string SQL_Server { get; set; }

// public string HTTP_User_Agent { get; set; }

// public string User_Agent_Browser_Name { get; set; }

// public Nullable<int> User_Agent_Browser_Major_Version { get; set; }

// public string User_Agent_OS_Name { get; set; }

// public string User_Agent_Device { get; set; }

// public string Request_Method { get; set; }

// public string Exception_Error_Message { get; set; }

// public string Stack_Trace { get; set; }

// public string Source_Context { get; set; }

// public Nullable<int> Exception_Data_Source_Key { get; set; }

// public string Exception_Data_Source_Name { get; set; }

// public string Exception_Column_Name { get; set; }

// public string Exception_SQL_Command { get; set; }

// public string Exception_Location { get; set; }

// public string Element_List { get; set; }

// public string Session_Info { get; set; }

// public string Session_Id { get; set; }
```

```
//     public string Setting_Group { get; set; }

//     public string Setting_Name { get; set; }

//     public string Missing_Image_Path { get; set; }

//     public string Node_Id { get; set; }

//     public Nullable<int> Thread_Id { get; set; }

//     public Nullable<long> Previous_Change_Version { get; set; }

// }

//}
```

Application_Event_w.Tags.cs (Monitoring.ETL.Domain\Model\Application_Event_w.Tags.cs):

```
//using System;

//using System.Collections.Generic;

//using System.Linq;

//using System.Text;

//using System.Threading.Tasks;

//using Monitoring.ETL.Domain.Warehouse;

//namespace Monitoring.ETL.Domain.Model

//{{

// public partial class Application_Event_w : IWarehouseLoadModel, ITaggable

// {

//     public IEnumerable<Tag_w> Tags { get; set; }

// }

// public partial class Tag_w : IWarehouseLoadModel

// {
```

```
// }
```

```
//}
```

ConnectionTools.cs (Monitoring.ETL.Domain\Model\ConnectionTools.cs):

```
using System;
```

```
using System.Configuration;
```

```
using System.Data.Entity;
```

```
using System.Data.Entity.Core.EntityClient;
```

```
using System.Data.SqlClient;
```

```
using System.Reflection;
```

```
namespace Monitoring.ETL.Domain.Model
```

```
{
```

```
    public static class ConnectionTools
```

```
    {
```

```
        // all params are optional
```

```
        public static T ChangeDatabase<T>(
```

```
            this T source,
```

```
            string initialCatalog = "",
```

```
            string dataSource = "",
```

```
            string userId = "",
```

```
            string password = "",
```

```
            bool integratedSecurity = true,
```

```
            string configConnectionStringName = "")
```

```
        where T : DbContext
```

```
        /* this would be used if the
```

```

* connectionString name varied from
* the base EF class name */

{

    try

    {

        // use the const name if it's not null, otherwise

        // using the convention of connection string = EF contextname

        // grab the type name and we're done

        var configNameEf = string.IsNullOrEmpty(configConnectionStringName)

            ? source.GetType().Name

            : configConnectionStringName;

        // add a reference to System.Configuration

        var entityCnxStringBuilder = new EntityConnectionStringBuilder

            (ConfigurationManager.ConnectionStrings[configNameEf].ConnectionString);

        // init the sqlbuilder with the full EF connectionstring cargo

        var sqlCnxStringBuilder = new SqlConnectionStringBuilder

            (entityCnxStringBuilder.ProviderConnectionString);

        // only populate parameters with values if added

        if (!string.IsNullOrEmpty(initialCatalog))

            sqlCnxStringBuilder.InitialCatalog = initialCatalog;

        if (!string.IsNullOrEmpty(dataSource))

            sqlCnxStringBuilder.DataSource = dataSource;

        if (!string.IsNullOrEmpty(userId))

```

```

        sqlCnxStringBuilder.UserID = userId;

        if (!string.IsNullOrEmpty(password))

            sqlCnxStringBuilder.Password = password;


        // set the integrated security status

        sqlCnxStringBuilder.IntegratedSecurity = integratedSecurity;

        // now flip the properties that were changed

        source.Database.Connection.ConnectionString

            = sqlCnxStringBuilder.ConnectionString;

    }

    catch (Exception ex)

    {

        // set log item if required

    }


    return source;

}

}

}

```

Load_State.cs (Monitoring.ETL.Domain\Model\Load_State.cs):

```

//-----

// <auto-generated>

//   This code was generated from a template.

//

//   Manual changes to this file may cause unexpected behavior in your application.

```

```
// Manual changes to this file will be overwritten if the code is regenerated.
```

```
// </auto-generated>
```

```
//-----
```

```
namespace Monitoring.ETL.Domain.Model
```

```
{
```

```
    using System;
```

```
    using System.Collections.Generic;
```

```
    public partial class Load_State
```

```
    {
```

```
        public int Load_State_Key { get; set; }
```

```
        public string Load_Name { get; set; }
```

```
        public Nullable<int> Last_Load_Key { get; set; }
```

```
        public Nullable<System.DateTime> Last_Load_Date { get; set; }
```

```
        public Nullable<long> Last_Load_Bigint_Key { get; set; }
```

```
    }
```

```
}
```

```
Meta_Warehouse.Context.cs (Monitoring.ETL.Domain\Model\Meta_Warehouse.Context.cs):
```

```
//-----
```

```
// <auto-generated>
```

```
// This code was generated from a template.
```

```
//
```

```
// Manual changes to this file may cause unexpected behavior in your application.
```

```
// Manual changes to this file will be overwritten if the code is regenerated.
```



```
// </auto-generated>
```

```
//-----
```

```
namespace Monitoring.ETL.Domain.Model
```

```
{
```

```
    using System;
```

```
    using System.Data.Entity;
```

```
    using System.Data.Entity.Infrastructure;
```

```
    using System.Data.Entity.Core.Objects;
```

```
    using System.Linq;
```

```
    public partial class Meta_WarehouseEntities : DbContext
```

```
    {
```

```
        public Meta_WarehouseEntities()
```

```
            : base("name=Meta_WarehouseEntities")
```

```
        {
```

```
        }
```

```
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
```

```
        {
```

```
            throw new UnintentionalCodeFirstException();
```

```
        }
```

```
        public virtual DbSet<Load_State> Load_State { get; set; }
```

```
//public virtual DbSet<Application_Event_w> Application_Event_w { get; set; }
```

```
//public virtual DbSet<Tag_w> Tag_w { get; set; }
```

```

public virtual int Fact_Error_Add(Nullable<int> warehouse_Load_Key)
{
    var warehouse_Load_KeyParameter = warehouse_Load_Key.HasValue ?
        new ObjectParameter("Warehouse_Load_Key", warehouse_Load_Key) :
        new ObjectParameter("Warehouse_Load_Key", typeof(int));

    return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("Fact_Error_Add",
warehouse_Load_KeyParameter);
}

public virtual int Warehouse_Load_Add(ObjectParameter warehouse_Load_Key)
{
    return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("Warehouse_Load_Add",
warehouse_Load_Key);
}

public virtual int Fact_Application_Event_Add(Nullable<int> warehouse_Load_Key)
{
    var warehouse_Load_KeyParameter = warehouse_Load_Key.HasValue ?
        new ObjectParameter("Warehouse_Load_Key", warehouse_Load_Key) :
        new ObjectParameter("Warehouse_Load_Key", typeof(int));

    return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("Fact_Application_Event_Add",
warehouse_Load_KeyParameter);
}

```

```

public virtual int Fact_Login_Add(Nullable<int> warehouse_Load_Key)
{
    var warehouse_Load_KeyParameter = warehouse_Load_Key.HasValue ?
        new ObjectParameter("Warehouse_Load_Key", warehouse_Load_Key) :
        new ObjectParameter("Warehouse_Load_Key", typeof(int));

    return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction("Fact_Login_Add",
warehouse_Load_KeyParameter);
}

public virtual int Warehouse_Load_Error_Update(
    Nullable<int> warehouse_Load_Key,
    Nullable<int> error_Number,
    string error_Message,
    Nullable<int> error_Severity = 16,
    Nullable<int> error_State = 1,
    string error_Procedure = "",
    Nullable<int> error_Line = 0)
{
    var warehouse_Load_KeyParameter = warehouse_Load_Key.HasValue ? new
ObjectParameter("Warehouse_Load_Key", warehouse_Load_Key) : new ObjectParameter("Warehouse_Load_Key",
typeof(int));

    var error_NumberParameter = error_Number.HasValue ? new ObjectParameter("Error_Number", error_Number) :
new ObjectParameter("Error_Number", typeof(int));

```

```
var error_MessageParameter = new ObjectParameter("Error_Message", error_Message);
```

```
var error_SeverityParameter = error_Severity.HasValue ? new ObjectParameter("Error_Severity", error_Severity) :  
new ObjectParameter("Error_Severity", typeof(int));
```

```
var error_StateParameter = error_State.HasValue ? new ObjectParameter("Error_State", error_State) : new  
ObjectParameter("Error_State", typeof(int));
```

```
var error_ProcedureParameter = new ObjectParameter("Error_Procedure", error_Procedure);
```

```
var error_LineParameter = error_Line.HasValue ? new ObjectParameter("Error_Line", error_Line) : new  
ObjectParameter("Error_Line", typeof(int));
```

```
return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction(  
  
    "Warehouse_Load_Error_Update",  
  
    warehouse_Load_KeyParameter,  
  
    error_NumberParameter,  
  
    error_MessageParameter,  
  
    error_SeverityParameter,  
  
    error_StateParameter,  
  
    error_ProcedureParameter,  
  
    error_LineParameter);  
  
}  
  
}
```

```
}
```

```
Meta_Warehouse.Context.tt (Monitoring.ETL.Domain\Model\Meta_Warehouse.Context.tt):
```

```
<#@ template language="C#" debug="false" hostspecific="true"#>
```

```
<#@ include file="EF6.Utility.CS.ttinclude"#><#@
```

```
output extension=".cs"#><#
```

```
const string inputFile = @"Meta_Warehouse.edmx";
```

```
var textTransform = DynamicTextTransformation.Create(this);
```

```
var code = new CodeGenerationTools(this);
```

```
var ef = new MetadataTools(this);
```

```
var typeMapper = new TypeMapper(code, ef, textTransform.Errors);
```

```
var loader = new EdmMetadataLoader(textTransform.Host, textTransform.Errors);
```

```
var itemCollection = loader.CreateEdmItemCollection(inputFile);
```

```
var modelNamespace = loader.GetModelNamespace(inputFile);
```

```
var codeStringGenerator = new CodeStringGenerator(code, typeMapper, ef);
```

```
var container = itemCollection.OfType<EntityContainer>().FirstOrDefault();
```

```
if (container == null)
```

```
{
```

```
    return string.Empty;
```

```
}
```

```
#>
```

```
//-----
```

```
// <auto-generated>
```

```
// <#=CodeGenerationTools.GetResourceString("Template_GeneratedCodeCommentLine1")#>
```

```
//

// <#=CodeGenerationTools.GetResourceString("Template_GeneratedCodeCommentLine2")#>

// <#=CodeGenerationTools.GetResourceString("Template_GeneratedCodeCommentLine3")#>

// </auto-generated>

//-----

<#

var codeNamespace = code.VsNamespaceSuggestion();

if (!String.IsNullOrEmpty(codeNamespace))

{

#>

namespace <#=code.EscapeNamespace(codeNamespace)#>

{

<#

    PushIndent("  ");

}

#>

using System;

using System.Data.Entity;

using System.Data.Entity.Infrastructure;

<#

if (container.FunctionImports.Any())

{

#>
```

```
using System.Data.Entity.Core.Objects;
```

```
using System.Linq;
```

```
<#
```

```
}
```

```
#>
```

```
<#=Accessibility.ForType(container)#> partial class <#=code.Escape(container)#> : DbContext
```

```
{
```

```
    public <#=code.Escape(container)#>()
```

```
    {
```

```
        : base("name=<#=container.Name#>")
```

```
<#
```

```
if (!loader.IsLazyLoadingEnabled(container))
```

```
{
```

```
#>
```

```
    this.Configuration.LazyLoadingEnabled = false;
```

```
<#
```

```
}
```

```
foreach (var entitySet in container.BaseEntitySets.OfType<EntitySet>())
```

```
{
```

```
    // Note: the DbSet members are defined below such that the getter and
```

```
    // setter always have the same accessibility as the DbSet definition
```

```
    if (Accessibility.ForReadOnlyProperty(entitySet) != "public")
```

```
    {
```

```
#>
```

```
<#=codeStringGenerator.DbSetInitializer(entitySet)#>
```

```
<#
```

```
}
```

```
}
```

```
#>
```

```
}
```

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
```

```
{
```

```
    throw new UnintentionalCodeFirstException();
```

```
}
```

```
<#
```

```
    foreach (var entitySet in container.BaseEntitySets.OfType<EntitySet>())
```

```
{
```

```
#>
```

```
<#=codeStringGenerator.DbSet(entitySet)#>
```

```
<#
```

```
}
```

```
    foreach (var edmFunction in container.FunctionImports)
```

```
{
```

```
        WriteFunctionImport(typeMapper, codeStringGenerator, edmFunction, modelNamespace, includeMergeOption:
```

```
false);
```

```
}
```

```
#>
```



```
}
```

```
<#
```

```
if (!String.IsNullOrEmpty(codeNamespace))
```

```
{
```

```
    PopIndent();
```

```
#>
```

```
}
```

```
<#
```

```
}
```

```
#>
```

```
<#+
```

```
private void WriteFunctionImport(TypeMapper typeMapper, CodeStringGenerator codeStringGenerator, EdmFunction  
edmFunction, string modelNamespace, bool includeMergeOption)
```

```
{
```

```
    if (typeMapper.IsComposable(edmFunction))
```

```
    {
```

```
#>
```

```
[DbFunction("<#=edmFunction.NamespaceName#>", "<#=edmFunction.Name#>")]
```

```
<#=codeStringGenerator.ComposableFunctionMethod(edmFunction, modelNamespace)#>
```

```
{
```

```
<#+
```

```
    codeStringGenerator.WriteFunctionParameters(edmFunction, WriteFunctionParameter);
```

```
#>
```

```

        <#codeStringGenerator.ComposableCreateQuery(edmFunction, modelNamespace)#>

    }

<#++

    }

    else

    {

#>

        <#codeStringGenerator.FunctionMethod(edmFunction, modelNamespace, includeMergeOption)#>

        {

<#++

            codeStringGenerator.WriteFunctionParameters(edmFunction, WriteFunctionParameter);

#>

            <#codeStringGenerator.ExecuteFunction(edmFunction, modelNamespace, includeMergeOption)#>

            }

<#++

            if (typeMapper.GenerateMergeOptionFunction(edmFunction, includeMergeOption))

            {

                WriteFunctionImport(typeMapper, codeStringGenerator, edmFunction, modelNamespace, includeMergeOption:

true);

            }

        }

    }

}

public void WriteFunctionParameter(string name, string isNotNull, string notNullInit, string nullInit)

{

```

```
#>
```

```
var <#=name#> = <#=isNotNull#> ?
```

```
<#=notNullInit#> :
```

```
<#=nullInit#>;
```

```
<#+
```

```
}
```

```
public const string TemplateId = "CSharp_DbContext_Context_EF6";
```

```
public class CodeStringGenerator
```

```
{
```

```
    private readonly CodeGenerationTools _code;
```

```
    private readonly TypeMapper _typeMapper;
```

```
    private readonly MetadataTools _ef;
```

```
    public CodeStringGenerator(CodeGenerationTools code, TypeMapper typeMapper, MetadataTools ef)
```

```
    {
```

```
        ArgumentNotNull(code, "code");
```

```
        ArgumentNotNull(typeMapper, "typeMapper");
```

```
        ArgumentNotNull(ef, "ef");
```

```
        _code = code;
```

```
        _typeMapper = typeMapper;
```

```
        _ef = ef;
```

```
    }
```

```

public string Property(EdmProperty edmProperty)
{
    return string.Format(
        CultureInfo.InvariantCulture,
        "{0} {1} {2} {{ {3}get; {4}set; }}",
        Accessibility.ForProperty(edmProperty),
        _typeMapper.GetTypeName(edmProperty.TypeUsage),
        _code.Escape(edmProperty),
        _code.SpaceAfter(Accessibility.ForGetter(edmProperty)),
        _code.SpaceAfter(Accessibility.ForSetter(edmProperty)));
}

```

```

public string NavigationProperty(NavigationProperty navProp)
{
    var endType = _typeMapper.GetTypeName(navProp.ToEndMember.GetEntityType());

    return string.Format(
        CultureInfo.InvariantCulture,
        "{0} {1} {2} {{ {3}get; {4}set; }}",
        AccessibilityAndVirtual(Accessibility.ForNavigationProperty(navProp)),
        navProp.ToEndMember.RelationshipMultiplicity == RelationshipMultiplicity.Many ? ("ICollection<" + endType +
">") : endType,
        _code.Escape(navProp),
        _code.SpaceAfter(Accessibility.ForGetter(navProp)),
        _code.SpaceAfter(Accessibility.ForSetter(navProp)));
}

```

```
public string AccessibilityAndVirtual(string accessibility)

{

    return accessibility + (accessibility != "private" ? " virtual" : "");

}
```

```
public string EntityClassOpening(EntityType entity)

{

    return string.Format(

        CultureInfo.InvariantCulture,

        "{0} {1}partial class {2}{3}",

        Accessibility.ForType(entity),

        _code.SpaceAfter(_code.AbstractOption(entity)),

        _code.Escape(entity),

        _code.StringBefore(" : ", _typeMapper.GetTypeName(entity.BaseType)));

}
```

```
public string EnumOpening(SimpleType enumType)

{

    return string.Format(

        CultureInfo.InvariantCulture,

        "{0} enum {1} : {2}",

        Accessibility.ForType(enumType),

        _code.Escape(enumType),

        _code.Escape(_typeMapper.UnderlyingClrType(enumType)));

}
```

```

public void WriteFunctionParameters(EdmFunction edmFunction, Action<string, string, string, string> writeParameter)
{
    var parameters = FunctionImportParameter.Create(edmFunction.Parameters, _code, _ef);

    foreach (var parameter in parameters.Where(p => p.NeedsLocalVariable))
    {
        var isNotNull = parameter.IsNullableOfT ? parameter.FunctionParameterName + ".HasValue" :
parameter.FunctionParameterName + " != null";

        var notNullInit = "new ObjectParameter(\"" + parameter.EsqlParameterName + "\", " +
parameter.FunctionParameterName + ")";

        var nullInit = "new ObjectParameter(\"" + parameter.EsqlParameterName + "\", typeof(" +
TypeMapper.FixNamespaces(parameter.RawClrTypeName) + "))";

        writeParameter(parameter.LocalVariableName, isNotNull, notNullInit, nullInit);
    }
}

```

```

public string ComposableFunctionMethod(EdmFunction edmFunction, string modelNamespace)
{
    var parameters = _typeMapper.GetParameters(edmFunction);

    return string.Format(
        CultureInfo.InvariantCulture,
        "{0} IQueryable<{1}> {2}({3})",
        AccessibilityAndVirtual(Accessibility.ForMethod(edmFunction)),
        _typeMapper.GetTypeName(_typeMapper.GetReturnType(edmFunction), modelNamespace),
        _code.Escape(edmFunction),

```

```

        string.Join(", ", parameters.Select(p => TypeMapper.FixNamespaces(p.FunctionParameterType) + " " +
p.FunctionParameterName).ToArray());
    }

```

```

public string ComposableCreateQuery(EdmFunction edmFunction, string modelNamespace)

```

```

{
    var parameters = _typeMapper.GetParameters(edmFunction);

    return string.Format(
        CultureInfo.InvariantCulture,
        "return ((IObjectContextAdapter)this).ObjectContext.CreateQuery<{0}>(\"[{1}].[{2}]({{3}})\",
        _typeMapper.GetTypeName(_typeMapper.GetReturnType(edmFunction), modelNamespace),
        edmFunction.NamespaceName,
        edmFunction.Name,
        string.Join(", ", parameters.Select(p => "@" + p.EsSqlParameterName).ToArray()),
        _code.StringBefore(", ", string.Join(", ", parameters.Select(p => p.ExecuteParameterName).ToArray())));
}

```

```

public string FunctionMethod(EdmFunction edmFunction, string modelNamespace, bool includeMergeOption)

```

```

{
    var parameters = _typeMapper.GetParameters(edmFunction);

    var returnType = _typeMapper.GetReturnType(edmFunction);

    var paramList = String.Join(", ", parameters.Select(p => TypeMapper.FixNamespaces(p.FunctionParameterType) +
" " + p.FunctionParameterName).ToArray());

    if (includeMergeOption)

```

```

{

    paramList = _code.StringAfter(paramList, ", ") + "MergeOption mergeOption";

}

return string.Format(

    CultureInfo.InvariantCulture,

    "{0} {1} {2}{3}",

    AccessibilityAndVirtual(Accessibility.ForMethod(edmFunction)),

    returnType == null ? "int" : "ObjectResult<" + _typeMapper.GetTypeName(returnType, modelNamespace) + ">",

    _code.Escape(edmFunction),

    paramList);

}

```

```

public string ExecuteFunction(EdmFunction edmFunction, string modelNamespace, bool includeMergeOption)

{

    var parameters = _typeMapper.GetParameters(edmFunction);

    var returnType = _typeMapper.GetReturnType(edmFunction);

    var callParams = _code.StringBefore(", ", String.Join(", ", parameters.Select(p =>

p.ExecuteParameterName).ToArray()));

    if (includeMergeOption)

    {

        callParams = ", mergeOption" + callParams;

    }

    return string.Format(

```



```

        CultureInfo.InvariantCulture,

        "return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction{0}(\\"{1}\\\"{2});",

        returnType == null ? "" : "<" + _typeMapper.GetTypeName(returnType, modelNamespace) + ">",

        edmFunction.Name,

        callParams);
    }

```

```

public string DbSet(EntitySet entitySet)
{
    return string.Format(

        CultureInfo.InvariantCulture,

        "{0} virtual DbSet<{1}> {2} {{ get; set; }}",

        Accessibility.ForReadOnlyProperty(entitySet),

        _typeMapper.GetTypeName(entitySet.ElementType),

        _code.Escape(entitySet));
}

```

```

public string DbSetInitializer(EntitySet entitySet)
{
    return string.Format(

        CultureInfo.InvariantCulture,

        "{0} = Set<{1}>();",

        _code.Escape(entitySet),

        _typeMapper.GetTypeName(entitySet.ElementType));
}

```

```

public string UsingDirectives(bool inHeader, bool includeCollections = true)
{
    return inHeader == string.IsNullOrEmpty(_code.VsNamespaceSuggestion())
        ? string.Format(
            CultureInfo.InvariantCulture,
            "{0}using System;{1}" +
            "{2}",
            inHeader ? Environment.NewLine : "",
            includeCollections ? (Environment.NewLine + "using System.Collections.Generic;") : "",
            inHeader ? "" : Environment.NewLine)
        : "";
}
}

```

```

public class TypeMapper
{
    private const string ExternalTypeNameAttributeName =
        @"http://schemas.microsoft.com/ado/2006/04/codegeneration:ExternalTypeName";

    private readonly System.Collections.IList _errors;

    private readonly CodeGenerationTools _code;

    private readonly MetadataTools _ef;

    public static string FixNamespaces(string typeName)
    {
        return typeName.Replace("System.Data.Spatial.", "System.Data.Entity.Spatial.");
    }
}

```

```
}
```

```
public TypeMapper(CodeGenerationTools code, MetadataTools ef, System.Collections.IList errors)
```

```
{
```

```
    ArgumentNotNull(code, "code");
```

```
    ArgumentNotNull(ef, "ef");
```

```
    ArgumentNotNull(errors, "errors");
```

```
    _code = code;
```

```
    _ef = ef;
```

```
    _errors = errors;
```

```
}
```

```
public string GetTypeName(TypeUsage typeUsage)
```

```
{
```

```
    return typeUsage == null ? null : GetTypeName(typeUsage.EdmType, _ef.IsNullable(typeUsage),
```

```
modelNamespace: null);
```

```
}
```

```
public string GetTypeName(EdmType edmType)
```

```
{
```

```
    return GetTypeName(edmType, isNullable: null, modelNamespace: null);
```

```
}
```

```
public string GetTypeName(TypeUsage typeUsage, string modelNamespace)
```

```
{
```

```

        return typeUsage == null ? null : GetTypeName(typeUsage.EdmType, _ef.IsNullable(typeUsage),
modelNamespace);

    }

    public string GetTypeName(EdmType edmType, string modelNamespace)

    {

        return GetTypeName(edmType, isNullable: null, modelNamespace: modelNamespace);

    }

    public string GetTypeName(EdmType edmType, bool? isNullable, string modelNamespace)

    {

        if (edmType == null)

        {

            return null;

        }

        var collectionType = edmType as CollectionType;

        if (collectionType != null)

        {

            return String.Format(CultureInfo.InvariantCulture, "ICollection<{0}>", GetTypeName(collectionType.TypeUsage,
modelNamespace));

        }

        var typeName = _code.Escape(edmType.MetadataProperties

            .Where(p => p.Name == ExternalTypeNameAttributeName)

            .Select(p => (string)p.Value)

```

```
.FirstOrDefault())
```

```
?? (modelNamespace != null && edmType.NamespaceName != modelNamespace ?
```

```
    _code.CreateFullName(_code.EscapeNamespace(edmType.NamespaceName), _code.Escape(edmType)) :
```

```
    _code.Escape(edmType));
```

```
if (edmType is StructuralType)
```

```
{
```

```
    return typeName;
```

```
}
```

```
if (edmType is SimpleType)
```

```
{
```

```
    var clrType = UnderlyingClrType(edmType);
```

```
    if (!IsEnumType(edmType))
```

```
{
```

```
    typeName = _code.Escape(clrType);
```

```
}
```

```
typeName = FixNamespaces(typeName);
```

```
return clrType.IsValueType && isNullable == true ?
```

```
    String.Format(CultureInfo.InvariantCulture, "Nullable<{0}>", typeName) :
```

```
    typeName;
```

```
}
```

```
throw new ArgumentException("edmType");
```

```
}
```

```
public Type UnderlyingClrType(EdmType edmType)
```

```
{
```

```
    ArgumentNotNull(edmType, "edmType");
```

```
    var primitiveType = edmType as PrimitiveType;
```

```
    if (primitiveType != null)
```

```
    {
```

```
        return primitiveType.ClrEquivalentType;
```

```
    }
```

```
    if (IsEnumType(edmType))
```

```
    {
```

```
        return GetEnumUnderlyingType(edmType).ClrEquivalentType;
```

```
    }
```

```
    return typeof(object);
```

```
}
```

```
public object GetEnumMemberValue(MetadataItem enumMember)
```

```
{
```

```
    ArgumentNotNull(enumMember, "enumMember");
```

```
    var valueProperty = enumMember.GetType().GetProperty("Value");
```

```
    return valueProperty == null ? null : valueProperty.GetValue(enumMember, null);
```

```
}
```

```
public string GetEnumMemberName(Metadataltem enumMember)
```

```
{
```

```
    ArgumentNotNull(enumMember, "enumMember");
```

```
    var nameProperty = enumMember.GetType().GetProperty("Name");
```

```
    return nameProperty == null ? null : (string)nameProperty.GetValue(enumMember, null);
```

```
}
```

```
public System.Collections.IEnumerable GetEnumMembers(EdmType enumType)
```

```
{
```

```
    ArgumentNotNull(enumType, "enumType");
```

```
    var membersProperty = enumType.GetType().GetProperty("Members");
```

```
    return membersProperty != null
```

```
        ? (System.Collections.IEnumerable)membersProperty.GetValue(enumType, null)
```

```
        : Enumerable.Empty<Metadataltem>();
```

```
}
```

```
public bool EnumIsFlags(EdmType enumType)
```

```
{
```

```
    ArgumentNotNull(enumType, "enumType");
```

```
    var isFlagsProperty = enumType.GetType().GetProperty("IsFlags");
```

```
    return isFlagsProperty != null && (bool)isFlagsProperty.GetValue(enumType, null);
```

```

}

public bool IsEnumType(GlobalItem edmType)

{
    ArgumentNotNull(edmType, "edmType");

    return edmType.GetType().Name == "EnumType";
}

public PrimitiveType GetEnumUnderlyingType(EdmType enumType)

{
    ArgumentNotNull(enumType, "enumType");

    return (PrimitiveType)enumType.GetType().GetProperty("UnderlyingType").GetValue(enumType, null);
}

public string CreateLiteral(object value)

{
    if (value == null || value.GetType() != typeof(TimeSpan))
    {
        return _code.CreateLiteral(value);
    }

    return string.Format(CultureInfo.InvariantCulture, "new TimeSpan({0})", ((TimeSpan)value).Ticks);
}

```



```

public bool VerifyCaseInsensitiveTypeUniqueness(IEnumerable<string> types, string sourceFile)
{
    ArgumentNotNull(types, "types");

    ArgumentNotNull(sourceFile, "sourceFile");

    var hash = new HashSet<string>(StringComparer.InvariantCultureIgnoreCase);

    if (types.Any(item => !hash.Add(item)))
    {
        _errors.Add(
            new CompilerError(sourceFile, -1, -1, "6023",
                String.Format(CultureInfo.CurrentCulture,
CodeGenerationTools.GetResourceString("Template_CaseInsensitiveTypeConflict"))));

        return false;
    }

    return true;
}

public IEnumerable<SimpleType> GetEnumItemsToGenerate(IEnumerable<GlobalItem> itemCollection)
{
    return GetItemsToGenerate<SimpleType>(itemCollection)
        .Where(e => IsEnumType(e));
}

public IEnumerable<T> GetItemsToGenerate<T>(IEnumerable<GlobalItem> itemCollection) where T: EdmType
{
    return itemCollection

```

```

        .OfType<T>()

        .Where(i => !i.MetadataProperties.Any(p => p.Name == ExternalTypeNameAttributeName))

        .OrderBy(i => i.Name);
    }

```

```

public IEnumerable<string> GetAllGlobalItems(IEnumerable<GlobalItem> itemCollection)
{
    return itemCollection

        .Where(i => i is EntityType || i is ComplexType || i is EntityContainer || IsEnumType(i))

        .Select(g => GetGlobalItemName(g));
}

```

```

public string GetGlobalItemName(GlobalItem item)
{
    if (item is EdmType)
    {
        return ((EdmType)item).Name;
    }

    else
    {
        return ((EntityContainer)item).Name;
    }
}

```

```

public IEnumerable<EdmProperty> GetSimpleProperties(EntityType type)
{

```

```
return type.Properties.Where(p => p.TypeUsage.EdmType is SimpleType && p.DeclaringType == type);  
}
```

```
public IEnumerable<EdmProperty> GetSimpleProperties(ComplexType type)  
{  
    return type.Properties.Where(p => p.TypeUsage.EdmType is SimpleType && p.DeclaringType == type);  
}
```

```
public IEnumerable<EdmProperty> GetComplexProperties(EntityType type)  
{  
    return type.Properties.Where(p => p.TypeUsage.EdmType is ComplexType && p.DeclaringType == type);  
}
```

```
public IEnumerable<EdmProperty> GetComplexProperties(ComplexType type)  
{  
    return type.Properties.Where(p => p.TypeUsage.EdmType is ComplexType && p.DeclaringType == type);  
}
```

```
public IEnumerable<EdmProperty> GetPropertiesWithDefaultValues(EntityType type)  
{  
    return type.Properties.Where(p => p.TypeUsage.EdmType is SimpleType && p.DeclaringType == type &&  
p.DefaultValue != null);  
}
```

```
public IEnumerable<EdmProperty> GetPropertiesWithDefaultValues(ComplexType type)  
{
```

```

        return type.Properties.Where(p => p.TypeUsage.EdmType is SimpleType && p.DeclaringType == type &&
p.DefaultValue != null);

    }

    public IEnumerable<NavigationProperty> GetNavigationProperties(EntityType type)

    {

        return type.NavigationProperties.Where(np => np.DeclaringType == type);

    }

    public IEnumerable<NavigationProperty> GetCollectionNavigationProperties(EntityType type)

    {

        return type.NavigationProperties.Where(np => np.DeclaringType == type &&
np.ToEndMember.RelationshipMultiplicity == RelationshipMultiplicity.Many);

    }

    public FunctionParameter GetReturnParameter(EdmFunction edmFunction)

    {

        ArgumentNotNull(edmFunction, "edmFunction");

        var returnParamsProperty = edmFunction.GetType().GetProperty("ReturnParameters");

        return returnParamsProperty == null

            ? edmFunction.ReturnParameter

            : ((IEnumerable<FunctionParameter>)returnParamsProperty.GetValue(edmFunction, null)).FirstOrDefault();

    }

    public bool IsComposable(EdmFunction edmFunction)

```

```

{
    ArgumentNotNull(edmFunction, "edmFunction");

    var isComposableProperty = edmFunction.GetType().GetProperty("IsComposableAttribute");
    return isComposableProperty != null && (bool)isComposableProperty.GetValue(edmFunction, null);
}

public IEnumerable<FunctionImportParameter> GetParameters(EdmFunction edmFunction)
{
    return FunctionImportParameter.Create(edmFunction.Parameters, _code, _ef);
}

public TypeUsage GetReturnType(EdmFunction edmFunction)
{
    var returnParam = GetReturnParameter(edmFunction);
    return returnParam == null ? null : _ef.GetElementType(returnParam.TypeUsage);
}

public bool GenerateMergeOptionFunction(EdmFunction edmFunction, bool includeMergeOption)
{
    var returnType = GetReturnType(edmFunction);
    return !includeMergeOption && returnType != null && returnType.EdmType.BuiltInTypeKind ==
BuiltInTypeKind.EntityType;
}
}

```

```
public static void ArgumentNotNull<T>(T arg, string name) where T : class
```

```
{  
  
    if (arg == null)  
  
    {  
  
        throw new ArgumentNullException(name);  
  
    }  
  
}  
  
#>
```

Meta_Warehouse.cs (Monitoring.ETL.Domain\Model\Meta_Warehouse.cs):

```
//-----  
  
// <auto-generated>  
  
//   This code was generated from a template.  
  
//  
  
//   Manual changes to this file may cause unexpected behavior in your application.  
  
//   Manual changes to this file will be overwritten if the code is regenerated.  
  
// </auto-generated>  
  
//-----
```

Meta_Warehouse.Designer.cs (Monitoring.ETL.Domain\Model\Meta_Warehouse.Designer.cs):

```
// T4 code generation is enabled for model  
  
'C:\Projects\Monitoring.ETL.Domain\Monitoring.ETL.Domain\Model\Meta_Warehouse.edmx'.  
  
// To enable legacy code generation, change the value of the 'Code Generation Strategy' designer  
  
// property to 'LegacyObjectContext'. This property is available in the Properties Window when the model  
  
// is open in the designer.
```

```
// If no context and entity classes have been generated, it may be because you created an empty model but  
  
// have not yet chosen which version of Entity Framework to use. To generate a context class and entity  
  
// classes for your model, open the model in the designer, right-click on the designer surface, and  
  
// select 'Update Model from Database...', 'Generate Database from Model...', or 'Add Code Generation  
  
// Item...'.  
  

```

Meta_Warehouse.edmx (Monitoring.ETL.Domain\Model\Meta_Warehouse.edmx):

```
<?xml version="1.0" encoding="utf-8"?>  
  
<edmx:Edmx Version="3.0" xmlns:edmx="http://schemas.microsoft.com/ado/2009/11/edmx">  
  
  <!-- EF Runtime content -->  
  
  <edmx:Runtime>  
  
    <!-- SSDL content -->  
  
    <edmx:StorageModels>  
  
      <Schema Namespace="Meta_WarehouseModel.Store" Provider="System.Data.SqlClient"  
ProviderManifestToken="2012" Alias="Self"  
xmlns:store="http://schemas.microsoft.com/ado/2007/12/edm/EntityStoreSchemaGenerator"  
xmlns:customannotation="http://schemas.microsoft.com/ado/2013/11/edm/customannotation"  
xmlns="http://schemas.microsoft.com/ado/2009/11/edm/ssdl">  
  
        <EntityType Name="Application_Event_w">  
  
          <Key>  
  
            <PropertyRef Name="Warehouse_Load_Key" />  
  
            <PropertyRef Name="Load_Event_Number" />  
  
          </Key>  
  
          <Property Name="Warehouse_Load_Key" Type="int" Nullable="false" />  
  
          <Property Name="Load_Event_Number" Type="int" Nullable="false" />  
  

```

<Property Name="Error_Key" Type="int" />

<Property Name="PUN" Type="int" />

<Property Name="PCN" Type="int" />

<Property Name="Event_Date" Type="datetime" Nullable="false" />

<Property Name="Message" Type="varchar(max)" />

<Property Name="Event_Type" Type="varchar(max)" />

<Property Name="Event_Severity" Type="varchar(max)" />

<Property Name="Data_Center" Type="varchar(max)" />

<Property Name="Environment" Type="varchar(max)" />

<Property Name="Service_Application_Name" Type="varchar(max)" />

<Property Name="Filename" Type="varchar(max)" />

<Property Name="Path_Info" Type="varchar(max)" />

<Property Name="HTTP_Referer" Type="varchar(max)" />

<Property Name="Referer_Path" Type="varchar(max)" />

<Property Name="Web_Server" Type="varchar(max)" />

<Property Name="SQL_Server" Type="varchar(max)" />

<Property Name="HTTP_User_Agent" Type="varchar(max)" />

<Property Name="User_Agent_Browser_Name" Type="varchar(max)" />

<Property Name="User_Agent_Browser_Major_Version" Type="int" />

<Property Name="User_Agent_OS_Name" Type="varchar(max)" />

<Property Name="User_Agent_Device" Type="varchar(max)" />

<Property Name="Request_Method" Type="varchar(max)" />

<Property Name="Exception_Error_Message" Type="varchar(max)" />

<Property Name="Stack_Trace" Type="varchar(max)" />

<Property Name="Source_Context" Type="varchar(max)" />

<Property Name="Exception_Data_Source_Key" Type="int" />

<Property Name="Exception_Data_Source_Name" Type="varchar(max)" />

<Property Name="Exception_Column_Name" Type="varchar(max)" />

<Property Name="Exception_SQL_Command" Type="varchar(max)" />

<Property Name="Exception_Location" Type="varchar(max)" />

<Property Name="Element_List" Type="varchar(max)" />

<Property Name="Session_Info" Type="varchar(max)" />

<Property Name="Session_Id" Type="varchar(max)" />

<Property Name="Setting_Group" Type="varchar(max)" />

<Property Name="Setting_Name" Type="varchar(max)" />

<Property Name="Missing_Image_Path" Type="varchar(max)" />

<Property Name="Node_Id" Type="varchar(max)" />

<Property Name="Thread_Id" Type="int" />

<Property Name="Previous_Change_Version" Type="bigint" />

</EntityType>

<EntityType Name="Tag_w">

<Key>

<PropertyRef Name="Warehouse_Load_Key" />

<PropertyRef Name="Elastic_Id" />

<PropertyRef Name="Load_Event_Number" />

</Key>

<Property Name="Warehouse_Load_Key" Type="int" Nullable="false" />

<Property Name="Elastic_Id" Type="varchar(max)" Nullable="false" />

<Property Name="Tag_Name" Type="varchar(max)" Nullable="false" />

<Property Name="Load_Event_Number" Type="int" Nullable="false" />

</EntityType>

<EntityType Name="Load_State">

<Key>

<PropertyRef Name="Load_State_Key" />

</Key>

<Property Name="Load_State_Key" Type="int" StoreGeneratedPattern="Identity" Nullable="false" />

<Property Name="Load_Name" Type="varchar" MaxLength="50" Nullable="false" />

<Property Name="Last_Load_Key" Type="int" />

<Property Name="Last_Load_Date" Type="datetime" />

<Property Name="Last_Load_Bigint_Key" Type="bigint" />

</EntityType>

<Function Name="Warehouse_Load_Error_Update" Aggregate="false" BuiltIn="false" NiladicFunction="false"

IsComposable="false" ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">

<Parameter Name="Warehouse_Load_Key" Type="int" Mode="In" />

<Parameter Name="Error_Number" Type="int" Mode="In" />

<Parameter Name="Error_Message" Type="varchar(max)" Mode="In" />

<Parameter Name="Error_Severity" Type="int" Mode="In" />

<Parameter Name="Error_State" Type="int" Mode="In" />

<Parameter Name="Error_Procedure" Type="varchar(max)" Mode="In" />

<Parameter Name="Error_Line" Type="int" Mode="In" />

</Function>

<Function Name="Fact_Application_Event_Add" Aggregate="false" BuiltIn="false" NiladicFunction="false"

IsComposable="false" ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">

<Parameter Name="Warehouse_Load_Key" Type="int" Mode="In" />

</Function>

<Function Name="Fact_Error_Add" Aggregate="false" BuiltIn="false" NiladicFunction="false"

IsComposable="false" ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">

<Parameter Name="Warehouse_Load_Key" Type="int" Mode="In" />

```

</Function>

<Function Name="Fact_Login_Add" Aggregate="false" BuiltIn="false" NiladicFunction="false"
IsComposable="false" ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">

    <Parameter Name="Warehouse_Load_Key" Type="int" Mode="In" />

</Function>

<Function Name="Warehouse_Load_Add" Aggregate="false" BuiltIn="false" NiladicFunction="false"
IsComposable="false" ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">

    <Parameter Name="Warehouse_Load_Key" Type="int" Mode="InOut" />

</Function>

<EntityContainer Name="Meta_WarehouseModelStoreContainer">

    <EntitySet Name="Application_Event_w" EntityType="Self.Application_Event_w" Schema="dbo"
store:Type="Tables" />

    <EntitySet Name="Load_State" EntityType="Self.Load_State" Schema="dbo" store:Type="Tables" />

    <EntitySet Name="Tag_w" EntityType="Self.Tag_w" Schema="dbo" store:Type="Tables" />

</EntityContainer>

</Schema></edmx:StorageModels>

<!-- CSDL content -->

<edmx:ConceptualModels>

    <Schema Namespace="Meta_WarehouseModel" Alias="Self" annotation:UseStrongSpatialTypes="false"
xmlns:annotation="http://schemas.microsoft.com/ado/2009/02/edm/annotation"
xmlns:customannotation="http://schemas.microsoft.com/ado/2013/11/edm/customannotation"
xmlns="http://schemas.microsoft.com/ado/2009/11/edm">

        <EntityType Name="Load_State">

            <Key>

                <PropertyRef Name="Load_State_Key" />

            </Key>

```

```
<Property Name="Load_State_Key" Type="Int32" Nullable="false" annotation:StoreGeneratedPattern="Identity"
/>

<Property Name="Load_Name" Type="String" MaxLength="50" FixedLength="false" Unicode="false"
Nullable="false" />

<Property Name="Last_Load_Key" Type="Int32" />

<Property Name="Last_Load_Date" Type="DateTime" Precision="3" />

<Property Name="Last_Load_Bigint_Key" Type="Int64" />

</EntityType>

<EntityContainer Name="Meta_WarehouseEntities" annotation:LazyLoadingEnabled="true">

  <EntitySet Name="Load_State" EntityType="Self.Load_State" />

  <FunctionImport Name="Fact_Error_Add">

    <Parameter Name="Warehouse_Load_Key" Mode="In" Type="Int32" />

  </FunctionImport>

  <FunctionImport Name="Warehouse_Load_Add">

    <Parameter Name="Warehouse_Load_Key" Mode="InOut" Type="Int32" />

  </FunctionImport>

  <EntitySet Name="Application_Event_w" EntityType="Meta_WarehouseModel.Application_Event_w" />

  <EntitySet Name="Tag_w" EntityType="Meta_WarehouseModel.Tag_w" />

  <FunctionImport Name="Fact_Application_Event_Add">

    <Parameter Name="Warehouse_Load_Key" Mode="In" Type="Int32" />

  </FunctionImport>

  <FunctionImport Name="Fact_Login_Add">

    <Parameter Name="Warehouse_Load_Key" Mode="In" Type="Int32" />

  </FunctionImport>

  <FunctionImport Name="Warehouse_Load_Error_Update">

    <Parameter Name="Warehouse_Load_Key" Type="Int32" Mode="In" />
```

```
<Parameter Name="Error_Number" Type="Int32" Mode="In" />

<Parameter Name="Error_Message" Type="String" Mode="In" />

<Parameter Name="Error_Severity" Type="Int32" Mode="In" />

<Parameter Name="Error_State" Type="Int32" Mode="In" />

<Parameter Name="Error_Procedure" Type="String" Mode="In" />

<Parameter Name="Error_Line" Type="Int32" Mode="In" />
```

```
</FunctionImport>
```

```
</EntityContainer>
```

```
<EntityType Name="Application_Event_w">
```

```
<Key>
```

```
<PropertyRef Name="Warehouse_Load_Key" />
```

```
<PropertyRef Name="Load_Event_Number" />
```

```
</Key>
```

```
<Property Name="Warehouse_Load_Key" Type="Int32" Nullable="false" />
```

```
<Property Name="Load_Event_Number" Type="Int32" Nullable="false" />
```

```
<Property Name="Error_Key" Type="Int32" />
```

```
<Property Name="PUN" Type="Int32" />
```

```
<Property Name="PCN" Type="Int32" />
```

```
<Property Name="Event_Date" Type="DateTime" Nullable="false" Precision="3" />
```

```
<Property Name="Message" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />
```

```
<Property Name="Event_Type" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />
```

```
<Property Name="Event_Severity" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />
```

```
<Property Name="Data_Center" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />
```

```
<Property Name="Environment" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />
```

```
<Property Name="Service_Application_Name" Type="String" MaxLength="Max" FixedLength="false"
```

```
Unicode="false" />
```

```
<Property Name="Filename" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="Path_Info" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="HTTP_Referer" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="Referer_Path" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="Web_Server" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="SQL_Server" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="HTTP_User_Agent" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="User_Agent_Browser_Name" Type="String" MaxLength="Max" FixedLength="false"
```

```
Unicode="false" />
```

```
<Property Name="User_Agent_Browser_Major_Version" Type="Int32" />

<Property Name="User_Agent_OS_Name" Type="String" MaxLength="Max" FixedLength="false" Unicode="false"
/>
```

```
<Property Name="User_Agent_Device" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="Request_Method" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="Exception_Error_Message" Type="String" MaxLength="Max" FixedLength="false"
```

```
Unicode="false" />
```

```
<Property Name="Stack_Trace" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="Source_Context" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="Exception_Data_Source_Key" Type="Int32" />

<Property Name="Exception_Data_Source_Name" Type="String" MaxLength="Max" FixedLength="false"
```

```
Unicode="false" />
```

```
<Property Name="Exception_Column_Name" Type="String" MaxLength="Max" FixedLength="false"
```

```
Unicode="false" />
```

```
<Property Name="Exception_SQL_Command" Type="String" MaxLength="Max" FixedLength="false"
```

```
Unicode="false" />
```

```
<Property Name="Exception_Location" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />
```

```

<Property Name="Element_List" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="Session_Info" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="Session_Id" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="Setting_Group" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="Setting_Name" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="Missing_Image_Path" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="Node_Id" Type="String" MaxLength="Max" FixedLength="false" Unicode="false" />

<Property Name="Thread_Id" Type="Int32" />

<Property Name="Previous_Change_Version" Type="Int64" />

</EntityType>

<EntityType Name="Tag_w">

  <Key>

    <PropertyRef Name="Warehouse_Load_Key" />

    <PropertyRef Name="Elastic_Id" />

    <PropertyRef Name="Load_Event_Number" />

  </Key>

  <Property Name="Warehouse_Load_Key" Type="Int32" Nullable="false" />

  <Property Name="Elastic_Id" Type="String" MaxLength="20" FixedLength="true" Unicode="false" Nullable="false"

/>

  <Property Name="Tag_Name" Type="String" MaxLength="100" FixedLength="true" Unicode="false"

Nullable="false" />

  <Property Name="Load_Event_Number" Type="Int32" Nullable="false" />

</EntityType>

</Schema>

</edmx:ConceptualModels>

<!-- C-S mapping content -->

```

```

<edmx:Mappings>

  <Mapping Space="C-S" xmlns="http://schemas.microsoft.com/ado/2009/11/mapping/cs">

    <EntityContainerMapping StorageEntityContainer="Meta_WarehouseModelStoreContainer"
CdmEntityContainer="Meta_WarehouseEntities">

      <EntitySetMapping Name="Load_State">

        <EntityTypeMapping TypeName="Meta_WarehouseModel.Load_State">

          <MappingFragment StoreEntitySet="Load_State">

            <ScalarProperty Name="Load_State_Key" ColumnName="Load_State_Key" />

            <ScalarProperty Name="Load_Name" ColumnName="Load_Name" />

            <ScalarProperty Name="Last_Load_Key" ColumnName="Last_Load_Key" />

            <ScalarProperty Name="Last_Load_Date" ColumnName="Last_Load_Date" />

            <ScalarProperty Name="Last_Load_Bigint_Key" ColumnName="Last_Load_Bigint_Key" />

          </MappingFragment>

        </EntityTypeMapping>

      </EntitySetMapping>

      <FunctionImportMapping FunctionImportName="Fact_Error_Add"
FunctionName="Meta_WarehouseModel.Store.Fact_Error_Add" />

      <FunctionImportMapping FunctionImportName="Warehouse_Load_Add"
FunctionName="Meta_WarehouseModel.Store.Warehouse_Load_Add" />

      <EntitySetMapping Name="Application_Event_w">

        <EntityTypeMapping TypeName="Meta_WarehouseModel.Application_Event_w">

          <MappingFragment StoreEntitySet="Application_Event_w">

            <ScalarProperty Name="Previous_Change_Version" ColumnName="Previous_Change_Version" />

            <ScalarProperty Name="Thread_Id" ColumnName="Thread_Id" />

            <ScalarProperty Name="Node_Id" ColumnName="Node_Id" />

            <ScalarProperty Name="Missing_Image_Path" ColumnName="Missing_Image_Path" />

```


<ScalarProperty Name="Setting_Name" ColumnName="Setting_Name" />
<ScalarProperty Name="Setting_Group" ColumnName="Setting_Group" />
<ScalarProperty Name="Session_Id" ColumnName="Session_Id" />
<ScalarProperty Name="Session_Info" ColumnName="Session_Info" />
<ScalarProperty Name="Element_List" ColumnName="Element_List" />
<ScalarProperty Name="Exception_Location" ColumnName="Exception_Location" />
<ScalarProperty Name="Exception_SQL_Command" ColumnName="Exception_SQL_Command" />
<ScalarProperty Name="Exception_Column_Name" ColumnName="Exception_Column_Name" />
<ScalarProperty Name="Exception_Data_Source_Name" ColumnName="Exception_Data_Source_Name" />
<ScalarProperty Name="Exception_Data_Source_Key" ColumnName="Exception_Data_Source_Key" />
<ScalarProperty Name="Source_Context" ColumnName="Source_Context" />
<ScalarProperty Name="Stack_Trace" ColumnName="Stack_Trace" />
<ScalarProperty Name="Exception_Error_Message" ColumnName="Exception_Error_Message" />
<ScalarProperty Name="Request_Method" ColumnName="Request_Method" />
<ScalarProperty Name="User_Agent_Device" ColumnName="User_Agent_Device" />
<ScalarProperty Name="User_Agent_OS_Name" ColumnName="User_Agent_OS_Name" />
<ScalarProperty Name="User_Agent_Browser_Major_Version"

ColumnName="User_Agent_Browser_Major_Version" />

<ScalarProperty Name="User_Agent_Browser_Name" ColumnName="User_Agent_Browser_Name" />
<ScalarProperty Name="HTTP_User_Agent" ColumnName="HTTP_User_Agent" />
<ScalarProperty Name="SQL_Server" ColumnName="SQL_Server" />
<ScalarProperty Name="Web_Server" ColumnName="Web_Server" />
<ScalarProperty Name="Referer_Path" ColumnName="Referer_Path" />
<ScalarProperty Name="HTTP_REFERER" ColumnName="HTTP_REFERER" />
<ScalarProperty Name="Path_Info" ColumnName="Path_Info" />
<ScalarProperty Name="Filename" ColumnName="Filename" />

```
<ScalarProperty Name="Service_Application_Name" ColumnName="Service_Application_Name" />

<ScalarProperty Name="Environment" ColumnName="Environment" />

<ScalarProperty Name="Data_Center" ColumnName="Data_Center" />

<ScalarProperty Name="Event_Severity" ColumnName="Event_Severity" />

<ScalarProperty Name="Event_Type" ColumnName="Event_Type" />

<ScalarProperty Name="Message" ColumnName="Message" />

<ScalarProperty Name="Event_Date" ColumnName="Event_Date" />

<ScalarProperty Name="PCN" ColumnName="PCN" />

<ScalarProperty Name="PUN" ColumnName="PUN" />

<ScalarProperty Name="Error_Key" ColumnName="Error_Key" />

<ScalarProperty Name="Load_Event_Number" ColumnName="Load_Event_Number" />

<ScalarProperty Name="Warehouse_Load_Key" ColumnName="Warehouse_Load_Key" />

</MappingFragment>

</EntityTypeMapping>

</EntitySetMapping>

<EntitySetMapping Name="Tag_w">

  <EntityTypeMapping TypeName="Meta_WarehouseModel.Tag_w">

    <MappingFragment StoreEntitySet="Tag_w">

      <ScalarProperty Name="Warehouse_Load_Key" ColumnName="Warehouse_Load_Key" />

      <ScalarProperty Name="Elastic_Id" ColumnName="Elastic_Id" />

      <ScalarProperty Name="Tag_Name" ColumnName="Tag_Name" />

      <ScalarProperty Name="Load_Event_Number" ColumnName="Load_Event_Number" />

    </MappingFragment>

  </EntityTypeMapping>

</EntitySetMapping>

<FunctionImportMapping FunctionImportName="Fact_Application_Event_Add"
```

FunctionName="Meta_WarehouseModel.Store.Fact_Application_Event_Add" />

<FunctionImportMapping FunctionImportName="Fact_Login_Add"

FunctionName="Meta_WarehouseModel.Store.Fact_Login_Add" />

<FunctionImportMapping FunctionImportName="Warehouse_Load_Error_Update"

FunctionName="Meta_WarehouseModel.Store.Warehouse_Load_Error_Update" />

</EntityContainerMapping>

</Mapping>

</edmx:Mappings>

</edmx:Runtime>

<!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->

<Designer xmlns="http://schemas.microsoft.com/ado/2009/11/edmx">

<Connection>

<DesignerInfoPropertySet>

<DesignerProperty Name="MetadataArtifactProcessing" Value="EmbedInOutputAssembly" />

</DesignerInfoPropertySet>

</Connection>

<Options>

<DesignerInfoPropertySet>

<DesignerProperty Name="ValidateOnBuild" Value="true" />

<DesignerProperty Name="EnablePluralization" Value="false" />

<DesignerProperty Name="IncludeForeignKeysInModel" Value="true" />

<DesignerProperty Name="UseLegacyProvider" Value="false" />

<DesignerProperty Name="CodeGenerationStrategy" Value="None" />

</DesignerInfoPropertySet>

</Options>

<!-- Diagram content (shape and connector positions) -->

<Diagrams></Diagrams>

</Designer>

</edmx:Edmx>

Meta_Warehouse.edmx.diagram (Monitoring.ETL.Domain\Model\Meta_Warehouse.edmx.diagram):

<?xml version="1.0" encoding="utf-8"?>

<edmx:Edmx Version="3.0" xmlns:edmx="http://schemas.microsoft.com/ado/2009/11/edmx">

<!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->

<edmx:Designer xmlns="http://schemas.microsoft.com/ado/2009/11/edmx">

<!-- Diagram content (shape and connector positions) -->

<edmx:Diagrams>

<Diagram DiagramId="90b610b9a28545b9946577cca6e053ee" Name="Diagram1">

<EntityTypeShape EntityType="Meta_WarehouseModel.Load_State" Width="1.5" PointX="0.75" PointY="0.75"

IsExpanded="true" />

<EntityTypeShape EntityType="Meta_WarehouseModel.Application_Event_w" Width="2.625" PointX="3.625"

PointY="0.5" />

<EntityTypeShape EntityType="Meta_WarehouseModel.Tag_w" Width="1.5" PointX="0.75" PointY="3.75" />

</Diagram>

</edmx:Diagrams>

</edmx:Designer>

</edmx:Edmx>

Meta_Warehouse.tt (Monitoring.ETL.Domain\Model\Meta_Warehouse.tt):

<#@ template language="C#" debug="false" hostspecific="true"#>

<#@ include file="EF6.Utility.CS.ttinclude"#><#@

output extension=".cs"#><#

```

const string inputFile = @"Meta_Warehouse.edmx";

var textTransform = DynamicTextTransformation.Create(this);

var code = new CodeGenerationTools(this);

var ef = new MetadataTools(this);

var typeMapper = new TypeMapper(code, ef, textTransform.Errors);

var fileManager = EntityFrameworkTemplateFileManager.Create(this);

var itemCollection = new EdmMetadataLoader(textTransform.Host,
textTransform.Errors).CreateEdmItemCollection(inputFile);

var codeStringGenerator = new CodeStringGenerator(code, typeMapper, ef);

if (!typeMapper.VerifyCaseInsensitiveTypeUniqueness(typeMapper.GetAllGlobalItems(itemCollection), inputFile))
{
    return string.Empty;
}

WriteHeader(codeStringGenerator, fileManager);

foreach (var entity in typeMapper.GetItemsToGenerate<EntityType>(itemCollection))
{
    fileManager.StartNewFile(entity.Name + ".cs");

    BeginNamespace(code);

    #>

    <#=codeStringGenerator.UsingDirectives(inHeader: false)#>

    <#=codeStringGenerator.EntityClassOpening(entity)#>

    {

```

```

<#

    var propertiesWithDefaultValues = typeMapper.GetPropertiesWithDefaultValues(entity);

    var collectionNavigationProperties = typeMapper.GetCollectionNavigationProperties(entity);

    var complexProperties = typeMapper.GetComplexProperties(entity);

    if (propertiesWithDefaultValues.Any() || collectionNavigationProperties.Any() || complexProperties.Any())
    {
#>

        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
"CA2214:DoNotCallOverridableMethodsInConstructors")]

        public <#=code.Escape(entity)#>()

        {
<#

            foreach (var edmProperty in propertiesWithDefaultValues)

            {
#>

                this.<#=code.Escape(edmProperty)#> = <#=typeMapper.CreateLiteral(edmProperty.DefaultValue)#>;

<#

            }

            foreach (var navigationProperty in collectionNavigationProperties)

            {
#>

                this.<#=code.Escape(navigationProperty)#> = new

HashSet<<#=typeMapper.GetTypeName(navigationProperty.ToEndMember.GetEntityType())#>>();

<#

```

```
}

foreach (var complexProperty in complexProperties)

{

#>

    this.<#=code.Escape(complexProperty)#> = new

<#=typeMapper.GetTypeName(complexProperty.TypeUsage)#>();

<#

    }

#>

}

<#

    }

var simpleProperties = typeMapper.GetSimpleProperties(entity);

if (simpleProperties.Any())

{

    foreach (var edmProperty in simpleProperties)

    {

#>

        <#=codeStringGenerator.Property(edmProperty)#>

<#

    }

}
```

```

        if (complexProperties.Any())
        {
#>

<#

        foreach(var complexProperty in complexProperties)
        {
#>

        <#=codeStringGenerator.Property(complexProperty)#>

<#

        }

    }

    var navigationProperties = typeMapper.GetNavigationProperties(entity);

    if (navigationProperties.Any())
    {
#>

<#

        foreach (var navigationProperty in navigationProperties)
        {
            if (navigationProperty.ToEndMember.RelationshipMultiplicity == RelationshipMultiplicity.Many)
            {
#>

            [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
"CA2227:CollectionPropertiesShouldBeReadOnly")]

```



```

<#
    }

#>

<#=codeStringGenerator.NavigationProperty(navigationProperty)#>

<#
    }

    }

#>

}

<#

    EndNamespace(code);

}

foreach (var complex in typeMapper.GetItemsToGenerate<ComplexType>(itemCollection))

{

    fileManager.StartNewFile(complex.Name + ".cs");

    BeginNamespace(code);

#>

<#=codeStringGenerator.UsingDirectives(inHeader: false, includeCollections: false)#>

<#=Accessibility.ForType(complex)#> partial class <#=code.Escape(complex)#>

{

<#

    var complexProperties = typeMapper.GetComplexProperties(complex);

    var propertiesWithDefaultValues = typeMapper.GetPropertiesWithDefaultValues(complex);

    if (propertiesWithDefaultValues.Any() || complexProperties.Any())

```

```

{
#>

    public <#=code.Escape(complex)#>()

    {
<#

        foreach (var edmProperty in propertiesWithDefaultValues)

        {
#>

            this.<#=code.Escape(edmProperty)#> = <#=typeMapper.CreateLiteral(edmProperty.DefaultValue)#>;

<#

        }

        foreach (var complexProperty in complexProperties)

        {
#>

            this.<#=code.Escape(complexProperty)#> = new

<#=typeMapper.GetTypeName(complexProperty.TypeUsage)#>();

<#

        }
#>

    }

<#

}

var simpleProperties = typeMapper.GetSimpleProperties(complex);

```

```
if (simpleProperties.Any())

{

    foreach(var edmProperty in simpleProperties)

    {

#>

        <#=codeStringGenerator.Property(edmProperty)#>

<#

        }

    }

if (complexProperties.Any())

{

#>

<#

    foreach(var edmProperty in complexProperties)

    {

#>

        <#=codeStringGenerator.Property(edmProperty)#>

<#

        }

    }

#>

}

<#

    EndNamespace(code);
```

```

}

foreach (var enumType in typeMapper.GetEnumItemsToGenerate(itemCollection))
{
    fileManager.StartNewFile(enumType.Name + ".cs");

    BeginNamespace(code);

#>

<#=codeStringGenerator.UsingDirectives(inHeader: false, includeCollections: false)#>

<#

    if (typeMapper.EnumIsFlags(enumType))

    {

#>

[Flags]

<#

    }

#>

<#=codeStringGenerator.EnumOpening(enumType)#>

{

<#

    var foundOne = false;

    foreach (MetadataItem member in typeMapper.GetEnumMembers(enumType))

    {

        foundOne = true;

#>

        <#=code.Escape(typeMapper.GetEnumMemberName(member))#> =

```

```
<#=typeMapper.GetEnumMemberValue(member)#>,
```

```
<#
```

```
}
```

```
if (foundOne)
```

```
{
```

```
    this.GenerationEnvironment.Remove(this.GenerationEnvironment.Length - 3, 1);
```

```
}
```

```
#>
```

```
}
```

```
<#
```

```
    EndNamespace(code);
```

```
}
```

```
fileManager.Process();
```

```
#>
```

```
<#+
```

```
public void WriteHeader(CodeStringGenerator codeStringGenerator, EntityFrameworkTemplateFileManager
```

```
fileManager)
```

```
{
```

```
    fileManager.StartHeader();
```

```
#>
```

```
//-----
```

```
// <auto-generated>
```

```
// <#=CodeGenerationTools.GetResourceString("Template_GeneratedCodeCommentLine1")#>
```

```
//
```

```
// <#=CodeGenerationTools.GetResourceString("Template_GeneratedCodeCommentLine2")#>
```

```
// <#=CodeGenerationTools.GetResourceString("Template_GeneratedCodeCommentLine3")#>
```

```
// </auto-generated>
```

```
//-----
```

```
<#=codeStringGenerator.UsingDirectives(inHeader: true)#>
```

```
<#+
```

```
    fileManager.EndBlock();
```

```
}
```

```
public void BeginNamespace(CodeGenerationTools code)
```

```
{
```

```
    var codeNamespace = code.VsNamespaceSuggestion();
```

```
    if (!String.IsNullOrEmpty(codeNamespace))
```

```
    {
```

```
#>
```

```
namespace <#=code.EscapeNamespace(codeNamespace)#>
```

```
{
```

```
<#+
```

```
    PushIndent("  ");
```

```
    }
```

```
}
```

```
public void EndNamespace(CodeGenerationTools code)
```

```
{
```

```

        if (!String.IsNullOrEmpty(code.VsNamespaceSuggestion()))
        {
            PopIndent();
        }
    }
}
}
}
}
}

```

```

public const string TemplateId = "CSharp_DbContext_Types_EF6";

```

```

public class CodeStringGenerator

```

```

{
    private readonly CodeGenerationTools _code;

    private readonly TypeMapper _typeMapper;

    private readonly MetadataTools _ef;

```

```

    public CodeStringGenerator(CodeGenerationTools code, TypeMapper typeMapper, MetadataTools ef)

```

```

    {
        ArgumentNotNull(code, "code");

        ArgumentNotNull(typeMapper, "typeMapper");

        ArgumentNotNull(ef, "ef");

```

```

        _code = code;

        _typeMapper = typeMapper;

        _ef = ef;

```

```
}
```

```
public string Property(EdmProperty edmProperty)
```

```
{
```

```
    return string.Format(
```

```
        CultureInfo.InvariantCulture,
```

```
        "{0} {1} {2} {{ {3}get; {4}set; }}",
```

```
        Accessibility.ForProperty(edmProperty),
```

```
        _typeMapper.GetTypeName(edmProperty.TypeUsage),
```

```
        _code.Escape(edmProperty),
```

```
        _code.SpaceAfter(Accessibility.ForGetter(edmProperty)),
```

```
        _code.SpaceAfter(Accessibility.ForSetter(edmProperty)));
```

```
}
```

```
public string NavigationProperty(NavigationProperty navProp)
```

```
{
```

```
    var endType = _typeMapper.GetTypeName(navProp.ToEndMember.GetEntityType());
```

```
    return string.Format(
```

```
        CultureInfo.InvariantCulture,
```

```
        "{0} {1} {2} {{ {3}get; {4}set; }}",
```

```
        AccessibilityAndVirtual(Accessibility.ForNavigationProperty(navProp)),
```

```
        navProp.ToEndMember.RelationshipMultiplicity == RelationshipMultiplicity.Many ? ("ICollection<" + endType +  
">") : endType,
```

```
        _code.Escape(navProp),
```

```
        _code.SpaceAfter(Accessibility.ForGetter(navProp)),
```

```
        _code.SpaceAfter(Accessibility.ForSetter(navProp)));
```



```
}
```

```
public string AccessibilityAndVirtual(string accessibility)
```

```
{
```

```
    return accessibility + (accessibility != "private" ? " virtual" : "");
```

```
}
```

```
public string EntityClassOpening(EntityType entity)
```

```
{
```

```
    return string.Format(
```

```
        CultureInfo.InvariantCulture,
```

```
        "{0} {1}partial class {2}{3}",
```

```
        Accessibility.ForType(entity),
```

```
        _code.SpaceAfter(_code.AbstractOption(entity)),
```

```
        _code.Escape(entity),
```

```
        _code.StringBefore(" : ", _typeMapper.GetTypeName(entity.BaseType)));
```

```
}
```

```
public string EnumOpening(SimpleType enumType)
```

```
{
```

```
    return string.Format(
```

```
        CultureInfo.InvariantCulture,
```

```
        "{0} enum {1} : {2}",
```

```
        Accessibility.ForType(enumType),
```

```
        _code.Escape(enumType),
```

```
        _code.Escape(_typeMapper.UnderlyingClrType(enumType)));
```

```
}
```

```
public void WriteFunctionParameters(EdmFunction edmFunction, Action<string, string, string, string> writeParameter)
{
    var parameters = FunctionImportParameter.Create(edmFunction.Parameters, _code, _ef);

    foreach (var parameter in parameters.Where(p => p.NeedsLocalVariable))
    {
        var isNotNull = parameter.IsNullableOfT ? parameter.FunctionParameterName + ".HasValue" :
parameter.FunctionParameterName + " != null";

        var notNullInit = "new ObjectParameter(\"" + parameter.EsqlParameterName + "\", " +
parameter.FunctionParameterName + ")";

        var nullInit = "new ObjectParameter(\"" + parameter.EsqlParameterName + "\", typeof(" +
TypeMapper.FixNamespaces(parameter.RawClrTypeName) + "))";

        writeParameter(parameter.LocalVariableName, isNotNull, notNullInit, nullInit);
    }
}
```

```
public string ComposableFunctionMethod(EdmFunction edmFunction, string modelNamespace)
{
    var parameters = _typeMapper.GetParameters(edmFunction);

    return string.Format(
        CultureInfo.InvariantCulture,
        "{0} IQueryable<{1}> {2}{{3}}",
        AccessibilityAndVirtual(Accessibility.ForMethod(edmFunction)),
        _typeMapper.GetTypeName(_typeMapper.GetReturnType(edmFunction), modelNamespace),
```

```

        _code.Escape(edmFunction),

        string.Join(", ", parameters.Select(p => TypeMapper.FixNamespaces(p.FunctionParameterType) + " " +
p.FunctionParameterName).ToArray()));

    }

```

```

public string ComposableCreateQuery(EdmFunction edmFunction, string modelNamespace)

```

```

{
    var parameters = _typeMapper.GetParameters(edmFunction);

    return string.Format(

        CultureInfo.InvariantCulture,

        "return ((IObjectContextAdapter)this).ObjectContext.CreateQuery<{0}>(\"{{1}}.{{2}}({{3}})\",{{4}});",

        _typeMapper.GetTypeName(_typeMapper.GetReturnType(edmFunction), modelNamespace),

        edmFunction.NamespaceName,

        edmFunction.Name,

        string.Join(", ", parameters.Select(p => "@" + p.EsSqlParameterName).ToArray()),

        _code.StringBefore(" ", string.Join(" ", parameters.Select(p => p.ExecuteParameterName).ToArray())));
}

```

```

public string FunctionMethod(EdmFunction edmFunction, string modelNamespace, bool includeMergeOption)

```

```

{
    var parameters = _typeMapper.GetParameters(edmFunction);

    var returnType = _typeMapper.GetReturnType(edmFunction);

    var paramList = String.Join(", ", parameters.Select(p => TypeMapper.FixNamespaces(p.FunctionParameterType) +
" " + p.FunctionParameterName).ToArray());
}

```

```
if (includeMergeOption)
```

```
{
```

```
    paramList = _code.StringAfter(paramList, ", ") + "MergeOption mergeOption";
```

```
}
```

```
return string.Format(
```

```
    CultureInfo.InvariantCulture,
```

```
    "{0} {1} {2}({3})",
```

```
    AccessibilityAndVirtual(Accessibility.ForMethod(edmFunction)),
```

```
    returnType == null ? "int" : "ObjectResult<" + _typeMapper.GetTypeName(returnType, modelNamespace) + ">",
```

```
    _code.Escape(edmFunction),
```

```
    paramList);
```

```
}
```

```
public string ExecuteFunction(EdmFunction edmFunction, string modelNamespace, bool includeMergeOption)
```

```
{
```

```
    var parameters = _typeMapper.GetParameters(edmFunction);
```

```
    var returnType = _typeMapper.GetReturnType(edmFunction);
```

```
    var callParams = _code.StringBefore(", ", String.Join(", ", parameters.Select(p =>
```

```
p.ExecuteParameterName).ToArray()));
```

```
    if (includeMergeOption)
```

```
{
```

```
        callParams = ", mergeOption" + callParams;
```

```
}
```

```

return string.Format(

    CultureInfo.InvariantCulture,

    "return ((IObjectContextAdapter)this).ObjectContext.ExecuteFunction{0}(\\"{1}\\\"{2});",

    returnType == null ? "" : "<" + _typeMapper.GetTypeName(returnType, modelNamespace) + ">",

    edmFunction.Name,

    callParams);

}

```

```

public string DbSet(EntitySet entitySet)

{

    return string.Format(

        CultureInfo.InvariantCulture,

        "{0} virtual DbSet<{1}> {2} {{ get; set; }}",

        Accessibility.ForReadOnlyProperty(entitySet),

        _typeMapper.GetTypeName(entitySet.ElementType),

        _code.Escape(entitySet));

}

```

```

public string UsingDirectives(bool inHeader, bool includeCollections = true)

{

    return inHeader == string.IsNullOrEmpty(_code.VsNamespaceSuggestion())

        ? string.Format(

            CultureInfo.InvariantCulture,

            "{0}using System;{1}" +

            "{2}",

            inHeader ? Environment.NewLine : "",


```

```

        includeCollections ? (Environment.NewLine + "using System.Collections.Generic;") : "",

        inHeader ? "" : Environment.NewLine)

    : "";

}

}

public class TypeMapper

{

    private const string ExternalTypeNameAttributeName =

@"http://schemas.microsoft.com/ado/2006/04/codegeneration:ExternalTypeName";

    private readonly System.Collections.IList _errors;

    private readonly CodeGenerationTools _code;

    private readonly MetadataTools _ef;

    public TypeMapper(CodeGenerationTools code, MetadataTools ef, System.Collections.IList errors)

    {

        ArgumentNotNull(code, "code");

        ArgumentNotNull(ef, "ef");

        ArgumentNotNull(errors, "errors");

        _code = code;

        _ef = ef;

        _errors = errors;

    }

```

```

public static string FixNamespaces(string typeName)

{

    return typeName.Replace("System.Data.Spatial.", "System.Data.Entity.Spatial.");

}


public string GetTypeName(TypeUsage typeUsage)

{

    return typeUsage == null ? null : GetTypeName(typeUsage.EdmType, _ef.IsNullable(typeUsage),

modelNameNamespace: null);

}


public string GetTypeName(EdmType edmType)

{

    return GetTypeName(edmType, isNullable: null, modelNameNamespace: null);

}


public string GetTypeName(TypeUsage typeUsage, string modelNameNamespace)

{

    return typeUsage == null ? null : GetTypeName(typeUsage.EdmType, _ef.IsNullable(typeUsage),

modelNameNamespace);

}


public string GetTypeName(EdmType edmType, string modelNameNamespace)

{

    return GetTypeName(edmType, isNullable: null, modelNameNamespace: modelNameNamespace);

}

```

```

public string GetTypeName(EdmType edmType, bool? isNullable, string modelNamespace)
{
    if (edmType == null)
    {
        return null;
    }

    var collectionType = edmType as CollectionType;

    if (collectionType != null)
    {
        return String.Format(CultureInfo.InvariantCulture, "ICollection<{0}>", GetTypeName(collectionType.TypeUsage,
modelNamespace));
    }

    var typeName = _code.Escape(edmType.MetadataProperties
        .Where(p => p.Name == ExternalTypeNameAttributeName)
        .Select(p => (string)p.Value)
        .FirstOrDefault())

    ?? (modelNamespace != null && edmType.NamespaceName != modelNamespace ?
        _code.CreateFullName(_code.EscapeNamespace(edmType.NamespaceName), _code.Escape(edmType)) :
        _code.Escape(edmType));

    if (edmType is StructuralType)
    {
        return typeName;
    }

```



```
}
```

```
if (edmType is SimpleType)
```

```
{
```

```
    var clrType = UnderlyingClrType(edmType);
```

```
    if (!IsEnumType(edmType))
```

```
    {
```

```
        typeName = _code.Escape(clrType);
```

```
    }
```

```
    typeName = FixNamespaces(typeName);
```

```
    return clrType.IsValueType && isNullable == true ?
```

```
        String.Format(CultureInfo.InvariantCulture, "Nullable<{0}>", typeName) :
```

```
        typeName;
```

```
}
```

```
throw new ArgumentException("edmType");
```

```
}
```

```
public Type UnderlyingClrType(EdmType edmType)
```

```
{
```

```
    ArgumentNotNull(edmType, "edmType");
```

```
    var primitiveType = edmType as PrimitiveType;
```

```
    if (primitiveType != null)
```

```

{
    return primitiveType.ClrEquivalentType;
}

if (IsEnumType(edmType))
{
    return GetEnumUnderlyingType(edmType).ClrEquivalentType;
}

return typeof(object);
}

public object GetEnumMemberValue(MetadatalItem enumMember)
{
    ArgumentNotNull(enumMember, "enumMember");

    var valueProperty = enumMember.GetType().GetProperty("Value");

    return valueProperty == null ? null : valueProperty.GetValue(enumMember, null);
}

public string GetEnumMemberName(MetadatalItem enumMember)
{
    ArgumentNotNull(enumMember, "enumMember");

    var nameProperty = enumMember.GetType().GetProperty("Name");

    return nameProperty == null ? null : (string)nameProperty.GetValue(enumMember, null);
}

```

```
}
```

```
public System.Collections.IEnumerable GetEnumMembers(EdmType enumType)
```

```
{
```

```
    ArgumentNotNull(enumType, "enumType");
```

```
    var membersProperty = enumType.GetType().GetProperty("Members");
```

```
    return membersProperty != null
```

```
        ? (System.Collections.IEnumerable)membersProperty.GetValue(enumType, null)
```

```
        : Enumerable.Empty<MetadataItem>();
```

```
}
```

```
public bool EnumIsFlags(EdmType enumType)
```

```
{
```

```
    ArgumentNotNull(enumType, "enumType");
```

```
    var isFlagsProperty = enumType.GetType().GetProperty("IsFlags");
```

```
    return isFlagsProperty != null && (bool)isFlagsProperty.GetValue(enumType, null);
```

```
}
```

```
public bool IsEnumType(GlobalItem edmType)
```

```
{
```

```
    ArgumentNotNull(edmType, "edmType");
```

```
    return edmType.GetType().Name == "EnumType";
```

```
}
```

```

public PrimitiveType GetEnumUnderlyingType(EdmType enumType)
{
    ArgumentNotNull(enumType, "enumType");

    return (PrimitiveType)enumType.GetType().GetProperty("UnderlyingType").GetValue(enumType, null);
}

```

```

public string CreateLiteral(object value)
{
    if (value == null || value.GetType() != typeof(TimeSpan))
    {
        return _code.CreateLiteral(value);
    }

    return string.Format(CultureInfo.InvariantCulture, "new TimeSpan({0})", ((TimeSpan)value).Ticks);
}

```

```

public bool VerifyCaseInsensitiveTypeUniqueness(IEnumerable<string> types, string sourceFile)
{
    ArgumentNotNull(types, "types");

    ArgumentNotNull(sourceFile, "sourceFile");

    var hash = new HashSet<string>(StringComparer.InvariantCultureIgnoreCase);

    if (types.Any(item => !hash.Add(item)))
    {

```

```

        _errors.Add(
            new CompilerError(sourceFile, -1, -1, "6023",
                String.Format(CultureInfo.CurrentCulture,
CodeGenerationTools.GetResourceString("Template_CaseInsensitiveTypeConflict"))));

        return false;
    }

    return true;
}

public IEnumerable<SimpleType> GetEnumItemsToGenerate(IEnumerable<GlobalItem> itemCollection)
{
    return GetItemsToGenerate<SimpleType>(itemCollection)
        .Where(e => IsEnumType(e));
}

public IEnumerable<T> GetItemsToGenerate<T>(IEnumerable<GlobalItem> itemCollection) where T: EdmType
{
    return itemCollection
        .OfType<T>()
        .Where(i => !i.MetadataProperties.Any(p => p.Name == ExternalTypeNameAttributeName))
        .OrderBy(i => i.Name);
}

public IEnumerable<string> GetAllGlobalItems(IEnumerable<GlobalItem> itemCollection)
{
    return itemCollection

```

```

        .Where(i => i is EntityType || i is ComplexType || i is EntityContainer || IsEnumType(i))

        .Select(g => GetGlobalItemName(g));
    }

```

```

public string GetGlobalItemName(GlobalItem item)
{
    if (item is EdmType)
    {
        return ((EdmType)item).Name;
    }

    else
    {
        return ((EntityContainer)item).Name;
    }
}

```

```

public IEnumerable<EdmProperty> GetSimpleProperties(EntityType type)
{
    return type.Properties.Where(p => p.TypeUsage.EdmType is SimpleType && p.DeclaringType == type);
}

```

```

public IEnumerable<EdmProperty> GetSimpleProperties(ComplexType type)
{
    return type.Properties.Where(p => p.TypeUsage.EdmType is SimpleType && p.DeclaringType == type);
}

```

```

public IEnumerable<EdmProperty> GetComplexProperties(EntityType type)

{

    return type.Properties.Where(p => p.TypeUsage.EdmType is ComplexType && p.DeclaringType == type);

}


public IEnumerable<EdmProperty> GetComplexProperties(ComplexType type)

{

    return type.Properties.Where(p => p.TypeUsage.EdmType is ComplexType && p.DeclaringType == type);

}


public IEnumerable<EdmProperty> GetPropertiesWithDefaultValues(EntityType type)

{

    return type.Properties.Where(p => p.TypeUsage.EdmType is SimpleType && p.DeclaringType == type &&
p.DefaultValue != null);

}


public IEnumerable<EdmProperty> GetPropertiesWithDefaultValues(ComplexType type)

{

    return type.Properties.Where(p => p.TypeUsage.EdmType is SimpleType && p.DeclaringType == type &&
p.DefaultValue != null);

}


public IEnumerable<NavigationProperty> GetNavigationProperties(EntityType type)

{

    return type.NavigationProperties.Where(np => np.DeclaringType == type);

}

```

```

public IEnumerable<NavigationProperty> GetCollectionNavigationProperties(EntityType type)
{
    return type.NavigationProperties.Where(np => np.DeclaringType == type &&
np.ToEndMember.RelationshipMultiplicity == RelationshipMultiplicity.Many);
}

public FunctionParameter GetReturnParameter(EdmFunction edmFunction)
{
    ArgumentNotNull(edmFunction, "edmFunction");

    var returnParamsProperty = edmFunction.GetType().GetProperty("ReturnParameters");

    return returnParamsProperty == null
        ? edmFunction.ReturnParameter
        : ((IEnumerable<FunctionParameter>)returnParamsProperty.GetValue(edmFunction, null)).FirstOrDefault();
}

public bool IsComposable(EdmFunction edmFunction)
{
    ArgumentNotNull(edmFunction, "edmFunction");

    var isComposableProperty = edmFunction.GetType().GetProperty("IsComposableAttribute");

    return isComposableProperty != null && (bool)isComposableProperty.GetValue(edmFunction, null);
}

public IEnumerable<FunctionImportParameter> GetParameters(EdmFunction edmFunction)

```



```

{

    return FunctionImportParameter.Create(edmFunction.Parameters, _code, _ef);

}


public TypeUsage GetReturnType(EdmFunction edmFunction)

{

    var returnParam = GetReturnParameter(edmFunction);

    return returnParam == null ? null : _ef.GetElementType(returnParam.TypeUsage);

}


public bool GenerateMergeOptionFunction(EdmFunction edmFunction, bool includeMergeOption)

{

    var returnType = GetReturnType(edmFunction);

    return !includeMergeOption && returnType != null && returnType.EdmType.BuiltInTypeKind ==

BuiltInTypeKind.EntityType;

}

}


public static void ArgumentNotNull<T>(T arg, string name) where T : class

{

    if (arg == null)

    {

        throw new ArgumentNullException(name);

    }

}

#>

```

Tag_w.cs (Monitoring.ETL.Domain\Model\Tag_w.cs):

////-----

/// <auto-generated>

/// This code was generated from a template.

///

/// Manual changes to this file may cause unexpected behavior in your application.

/// Manual changes to this file will be overwritten if the code is regenerated.

/// </auto-generated>

////-----

//namespace Monitoring.ETL.Domain.Model

//{

// using System;

// using System.Collections.Generic;

// public partial class Tag_w

// {

// public int Warehouse_Load_Key { get; set; }

// public string Elastic_Id { get; set; }

// public string Tag_Name { get; set; }

// public int Load_Event_Number { get; set; }

// }

//}

Monitoring.ETL.Domain.csproj (Monitoring.ETL.Domain\Monitoring.ETL.Domain.csproj):

<?xml version="1.0" encoding="utf-8"?>

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>

    <TargetFrameworks>net462</TargetFrameworks>

    <Title>Monitoring.ETL.Domain</Title>

    <Version>3.0.2</Version>

    <Copyright>Copyright 2023</Copyright>

    <Company>Plex Systems, Inc.</Company>

    <AssemblyVersion>3.0.2</AssemblyVersion>

    <FileVersion>3.0.2</FileVersion>

    <Description>Domain logic for ETL transfers</Description>

    <Configurations>Debug;Release;Nuget</Configurations>

    <GeneratePackageOnBuild>>false</GeneratePackageOnBuild>

    <PlatformTarget>AnyCPU</PlatformTarget>

    <AutoGenerateBindingRedirects>True</AutoGenerateBindingRedirects>

  </PropertyGroup>

  <PropertyGroup Condition=" '$(Configuration)|$(TargetFramework)|$(Platform)'=='Release|net462|AnyCPU' ">

    <DebugType>full</DebugType>

    <DebugSymbols>>true</DebugSymbols>

    <DocumentationFile></DocumentationFile>

    <OutputPath>bin\Release\</OutputPath>

    <Optimize>False</Optimize>

  </PropertyGroup>

  <PropertyGroup Condition=" '$(Configuration)|$(TargetFramework)|$(Platform)'=='Debug|net462|AnyCPU' ">

    <DebugType>full</DebugType>

    <DebugSymbols>>true</DebugSymbols>

    <OutputPath>bin\Debug\</OutputPath>
```

```
<PlatformTarget>x64</PlatformTarget>

</PropertyGroup>

<PropertyGroup Condition="'$(Configuration)|$(TargetFramework)|$(Platform)'=='Nuget|net462|AnyCPU'">

  <DebugType>pdbonly</DebugType>

  <DebugSymbols>>true</DebugSymbols>

</PropertyGroup>

<ItemGroup>

  <Compile Remove="RabbitMQ\ODBC\*" />

  <EmbeddedResource Remove="RabbitMQ\ODBC\*" />

  <None Remove="RabbitMQ\ODBC\*" />

</ItemGroup>

<ItemGroup>

  <Compile Remove="Jira\JiraIssueLoadModel.cs" />

</ItemGroup>

<ItemGroup>

  <PackageReference Include="Atlassian.SDK" Version="13.0.0" />

  <PackageReference Include="EntityFramework" Version="6.4.4" />

  <PackageReference Include="Libraries.Common.Exceptions" Version="1.1.0" />

  <PackageReference Include="Microsoft.Azure.Storage.Queue" Version="11.2.3" />

  <PackageReference Include="Microsoft.SqlServer.DacFx.x64" Version="150.5282.3" />

  <PackageReference Include="Monitoring.Email" Version="4.1.4" />

  <PackageReference Include="Monitoring.ETL.Process" Version="3.0.0" />

  <PackageReference Include="Monitoring.UserExtensions" Version="2.0.0" />

  <PackageReference Include="Newtonsoft.Json" Version="13.0.1" />

  <PackageReference Include="Plex.Infrastructure.ConfigSystems" Version="1.1.1" />

  <PackageReference Include="RabbitMQ.Client" Version="5.1.2" />


```

```
<PackageReference Include="SonarQube.Net" Version="1.0.5" />

<PackageReference Include="System.Configuration.ConfigurationManager" Version="7.0.0" />

<PackageReference Include="Unofficial.Microsoft.SqlServer.SMO.2014" Version="12.0.2000.8" />

</ItemGroup>

<ItemGroup>

    <Compile Update="Model\Load_State.cs" AutoGen="True" DesignTime="True"
DependentUpon="Meta_Warehouse.tt" />

    <Compile Update="Model\Meta_Warehouse.Context.cs" AutoGen="True" DesignTime="True"
DependentUpon="Meta_Warehouse.Context.tt" />

    <Compile Update="Model\Meta_Warehouse.cs" AutoGen="True" DesignTime="True"
DependentUpon="Meta_Warehouse.tt" />

    <Compile Update="Model\Application_Event_w.cs" AutoGen="True" DesignTime="True"
DependentUpon="Meta_Warehouse.tt" />

    <Compile Update="Model\Meta_Warehouse.Designer.cs">

        <DesignTime>True</DesignTime>

        <AutoGen>True</AutoGen>

        <DependentUpon>Meta_Warehouse.edmx</DependentUpon>

    </Compile>

    <Compile Update="Model\Tag_w.cs" AutoGen="True" DesignTime="True" DependentUpon="Meta_Warehouse.tt" />

    <Compile Update="Warehouse\Model\Meta_Warehouse.WorkTables.cs">

        <DesignTime>True</DesignTime>

        <AutoGen>True</AutoGen>

        <DependentUpon>Meta_Warehouse.WorkTables.tt</DependentUpon>

    </Compile>

</ItemGroup>

<ItemGroup>
```

```
<EntityDeploy Include="Model\Meta_Warehouse.edmx">

  <Generator>EntityModelCodeGenerator</Generator>

  <LastGenOutput>Meta_Warehouse.Designer.cs</LastGenOutput>

</EntityDeploy>

<None Include="Model\Meta_Warehouse.edmx.diagram" DependentUpon="Meta_Warehouse.edmx" />

</ItemGroup>

<!--<ItemGroup>

  <Content Include="Model\Meta_Warehouse.Context.tt">

    <Generator>TextTemplatingFileGenerator</Generator>

    <DependentUpon>Meta_Warehouse.edmx</DependentUpon>

    <LastGenOutput>Meta_Warehouse.Context.cs</LastGenOutput>

  </Content>

  <Content Include="Model\Meta_Warehouse.tt">

    <Generator>TextTemplatingFileGenerator</Generator>

    <DependentUpon>Meta_Warehouse.edmx</DependentUpon>

    <LastGenOutput>Meta_Warehouse.cs</LastGenOutput>

  </Content>

</ItemGroup>-->

<ItemGroup>

  <Service Include="{508349b6-6b84-4df5-91f0-309beebad82d}" />

</ItemGroup>

<ItemGroup>

  <None Update="Model\Meta_Warehouse.Context.tt">

    <Generator>TextTemplatingFileGenerator</Generator>

    <LastGenOutput>Meta_Warehouse.Context.cs</LastGenOutput>

  </None>
```

```
<None Update="Model\Meta_Warehouse.tt">

  <Generator>TextTemplatingFileGenerator</Generator>

  <LastGenOutput>Meta_Warehouse.cs</LastGenOutput>

</None>

<None Update="PLEX.snc">

  <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>

</None>

<None Update="Warehouse\Model\Meta_Warehouse.WorkTables.tt">

  <Generator>TextTemplatingFileGenerator</Generator>

  <LastGenOutput>Meta_Warehouse.WorkTables.cs</LastGenOutput>

</None>

</ItemGroup>

<ItemGroup>

  <Reference Include="Microsoft.CSharp" />

  <Reference Include="System.Management" />

  <Reference Include="System.Net.Http" />

  <Reference Include="System.Runtime.Caching" />

  <Reference Include="System.Web.Extensions" />

</ItemGroup>

</Project>
```

nuget.nuspec (Monitoring.ETL.Domain\nuget.nuspec):

```
<?xml version="1.0" encoding="utf-8"?>

<package xmlns="http://schemas.microsoft.com/packaging/2012/06/nuspec.xsd">

  <metadata>
```

<id>Monitoring.ETL.Domain</id>

<version>3.0.2</version>

<title>Monitoring.ETL.Domain</title>

<authors>Plex Monitoring Team</authors>

<owners>Plex Monitoring Team</owners>

<requireLicenseAcceptance>>false</requireLicenseAcceptance>

<description>ETL Domain Logic for the Monitoring Team ETL Services</description>

<summary>ETL Domain Logic for the Monitoring Team ETL Services</summary>

<copyright>Copyright 2023</copyright>

<dependencies>

<group targetFramework=".NETFramework4.6.2">

<dependency id="EntityFramework" version="6.4.4" />

<dependency id="Libraries.Common.Exceptions" version="1.1.0" />

<dependency id="Microsoft.Azure.Storage.Queue" version="11.2.3" />

<dependency id="Microsoft.SqlServer.ConnectionInfo.dll" version="1.0.1" />

<dependency id="Microsoft.SqlServer.DacFx.x64" version="150.5282.3" />

<dependency id="Microsoft.SqlServer.Management.Sdk.Sfc.dll" version="1.0.1" />

<dependency id="Microsoft.SqlServer.Smo.dll" version="1.0.1" />

<dependency id="Monitoring.Email" version="4.1.4" />

<dependency id="Monitoring.ETL.Process" version="3.0.0" />

<dependency id="Newtonsoft.Json" version="13.0.1" />

<dependency id="Plex.Infrastructure.ConfigSystems" version="1.1.1" />

<dependency id="RabbitMQ.Client" version="5.1.2" />

<dependency id="System.Configuration.ConfigurationManager" version="7.0.0" />

</group>


```
</dependencies>
```

```
</metadata>
```

```
<files>
```

```
<file src="bin\release\net462\Monitoring.ETL.Domain.dll" target="lib\net462\Monitoring.ETL.Domain.dll" />
```

```
<file src="bin\release\net462\Monitoring.ETL.Domain.pdb" target="lib\net462\Monitoring.ETL.Domain.pdb" />
```

```
</files>
```

```
</package>
```

DelayFactory.cs (Monitoring.ETL.Domain\ODBC\DelayFactory.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Text;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.ODBC
```

```
{
```

```
    class DelayFactory
```

```
    {
```

```
    }
```

```
}
```

Extractor.cs (Monitoring.ETL.Domain\ODBC\Extractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Text;

using System.Threading.Tasks;


namespace Monitoring.ETL.Domain.ODBC
{

    class Extractor

    {

    }

}
```

RabbitMQTransformer.cs (Monitoring.ETL.Domain\ODBC\RabbitMQTransformer.cs):

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;


namespace Monitoring.ETL.Domain.ODBC
{

    public class RabbitMQTransformer : ITransformer<Statement, RabbitMQ.Model.OdbcStatement>

    {

        public RabbitMQTransformer()

        {
```

```
}
```

```
public async Task<IEnumerable<RabbitMQ.Model.OdbcStatement>> TransformAsync(
    IEnumerable<Statement> extracted,
    CancellationToken cancellationToken,
    IEtlProcessLogger logger)
{
    var results = await Task
        .WhenAll(extracted.Select(statement =>
            TransformAsync(statement, cancellationToken, logger)));
    return results.SelectMany(a => a);
}
```

```
private async Task<IEnumerable<RabbitMQ.Model.OdbcStatement>> TransformAsync(
    Statement statement,
    CancellationToken cancellationToken,
    IEtlProcessLogger logger)
{
    await Task.Yield();

    var items = new List<RabbitMQ.Model.OdbcStatement>();

    items.Add(new RabbitMQ.Model.OdbcStatement
    {
        ODBC_Session_Key = statement.ODBC_Session_Key,
        ODBC_Session_Statement_Key = statement.ODBC_Session_Statement_Key,
```

```

PCN = statement.PCN,

PUN = statement.PUN,

Session_Start_Date = statement.Session_Start_Date,

Statement_Start_Date = statement.Statement_Start_Date,

Equipment_Key = statement.Equipment_Key,

Client_Address = statement.Client_Address,

Custom_Property_List = statement.Custom_Property_List,

Machine_Name = statement.Machine_Name,

Success = statement.Success,

Column_Count = statement.Column_Count,

Row_Count = statement.Row_Count,

Byte_Count = statement.Byte_Count,

SQL_Server_Name = statement.Server_Name,

SQL_Statement = statement.SQL_Statement

});

return items;

}

}

}

```

Repository.cs (Monitoring.ETL.Domain\ODBC\Repository.cs):

```

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Threading;

```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.ODBC
```

```
{
```

```
    internal class Repository
```

```
    {
```

```
        public async Task<List<Statement>> GetAllAfterDate(DateTime date, CancellationToken cancellationToken)
```

```
        {
```

```
            var statements = new List<Statement>();
```

```
            //using (var connection = new SqlConnection(@"data source=AH_N1_NRT1\AH_N1_NRT1;User
```

```
Id=PLEX_SIEM_READ_PDS;Password='2n5d;&r~VUEVN6';initial catalog=Development;"))
```

```
            using (var connection = new SqlConnection(@"data source=AH_N1_NRT1\AH_N1_NRT1;Integrated
```

```
Security=true;initial catalog=Report_Services;"))
```

```
            using (var command = connection.CreateCommand())
```

```
            {
```

```
                var keyParam = command.CreateParameter();
```

```
                keyParam.ParameterName = "@Last_Key";
```

```
                keyParam.DbType = System.Data.DbType.Int64;
```

```
                keyParam.Direction = System.Data.ParameterDirection.Input;
```

```
                keyParam.Value = date;
```

```
                command.CommandType = System.Data.CommandType.Text;
```

```
                command.CommandText = @"
```

```
USE Report_Services;
```

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

SELECT TOP(1000)

@@SERVERNAME AS Server_Name,

S.PCN,

S.ODBC_Session_Key,

S.PUN,

S.Equipment_Key,

S.Client_Address,

S.Client_Application,

S.Custom_Property_List,

S.Session_Start_Date,

S.Machine_Name,

SS.ODBC_Session_Statement_Key,

SS.Statement_Start_Date,

SS.Success,

SS.Column_Count,

SS.Row_Count,

SS.Byte_Count,

SSS.SQL_Statement

FROM dbo.ODBC_Session AS S

JOIN dbo.ODBC_Session_Statement AS SS

ON SS.PCN = S.PCN

AND SS.ODBC_Session_Key = S.ODBC_Session_Key

JOIN dbo.ODBC_Session_Statement_SQL AS SSS

ON SSS.PCN = SS.PCN

```
AND SSS.ODBC_Session_Statement_Key = SS.ODBC_Session_Statement_Key
```

```
WHERE SS.ODBC_Session_Statement_Key > @Last_Key
```

```
ORDER BY
```

```
SS.ODBC_Session_Statement_Key";
```

```
command.Parameters.Add(keyParam);
```

```
await connection.OpenAsync();
```

```
var reader = await command.ExecuteReaderAsync();
```

```
if (reader.HasRows)
```

```
{
```

```
var columns = new Dictionary<string, int>();
```

```
for (int c = 0; c < reader.FieldCount; c++)
```

```
{
```

```
columns[reader.GetName(c)] = c;
```

```
}
```

```
while (await reader.ReadAsync(cancellationToken))
```

```
{
```

```
statements.Add(new Statement
```

```
{
```

```
Server_Name = await Get<string>(reader, columns["Server_Name"]),
```

```
PCN = await Get<int>(reader, columns["PCN"]),
```

```
ODBC_Session_Key = await Get<long>(reader, columns["ODBC_Session_Key"]),
```

```

PUN = await Get<int>(reader, columns["PCN"]),

Equipment_Key = await Get<int>(reader, columns["Equipment_Key"]),

Client_Address = await Get<string>(reader, columns["Client_Address"]),

Client_Application = await Get<string>(reader, columns["Client_Application"]),

Custom_Property_List = await Get<string>(reader, columns["Custom_Property_List"]),

Session_Start_Date = await Get<DateTime>(reader, columns["Session_Start_Date"]),

Machine_Name = await Get<string>(reader, columns["Machine_Name"]),

ODBC_Session_Statement_Key = await Get<long>(reader, columns["ODBC_Session_Statement_Key"]),

Statement_Start_Date = await Get<DateTime>(reader, columns["Statement_Start_Date"]),

Success = await Get<bool>(reader, columns["Success"]),

Column_Count = await Get<int>(reader, columns["Column_Count"]),

Row_Count = await Get<int>(reader, columns["Row_Count"]),

Byte_Count = await Get<int>(reader, columns["Byte_Count"]),

SQL_Statement = await Get<string>(reader, columns["SQL_Statement"])

});

}

}

connection.Close();

}

return statements;

}

private async Task<T> Get<T>(SqlDataReader reader, int ordinal)

{

```



```

var value = reader.GetValue(ordinal);

try

{

    return (await reader.IsDBNullAsync(ordinal)) ? default(T) : (T)reader.GetValue(ordinal);

}

catch (Exception ex)

{

    return default(T);

}

}

}

```

Statement.cs (Monitoring.ETL.Domain\ODBC\Statement.cs):

```
using System;
```

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.ODBC
```

```
{
```

```
    public class Statement : IExtractModel
```

```
    {
```

```
        public string Server_Name { get; internal set; }
```

```
        public int PCN { get; internal set; }
```

```
        public long ODBC_Session_Key { get; internal set; }
```

```
        public int PUN { get; internal set; }
```

```
        public int Equipment_Key { get; internal set; }
```

```

public long ODBC_Session_Statement_Key { get; internal set; }

public string Client_Address { get; internal set; }

public string Client_Application { get; internal set; }

public string Custom_Property_List { get; internal set; }

public DateTime Session_Start_Date { get; internal set; }

public string Machine_Name { get; internal set; }

public DateTime Statement_Start_Date { get; internal set; }

public bool Success { get; internal set; }

public int Column_Count { get; internal set; }

public int Row_Count { get; internal set; }

public int Byte_Count { get; internal set; }

public string SQL_Statement { get; internal set; }

}

}

```

PeriodicExtractionDelayFactory.cs (Monitoring.ETL.Domain\PeriodicExtractionDelayFactory.cs):

```

using System;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain

{

    /// <summary>

    /// Extraction Delay Factory that always sleeps for the specified time span regardless of the number of results.

```

```
/// </summary>
```

```
/// <typeparam name="T">The extract model type.</typeparam>
```

```
public abstract class PeriodicExtractionDelayFactory<T> : IExtractionDelayFactory<T>
```

```
{
```

```
    /// <summary>
```

```
    /// The length of time to delay.
```

```
    /// </summary>
```

```
    protected abstract TimeSpan DelayTimeSpan { get; }
```

```
    public async Task Create(ResultSet<T> results, CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
    {
```

```
        logger.Information($"Sleeping for {DelayTimeSpan.TotalSeconds} seconds.");
```

```
        await Task.Delay(DelayTimeSpan, cancellationToken);
```

```
    }
```

```
}
```

```
}
```

FolderProfile.pubxml (Monitoring.ETL.Domain\Properties\PublishProfiles\FolderProfile.pubxml):

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<!--
```

```
https://go.microsoft.com/fwlink/?LinkID=208121.
```

```
-->
```

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
```

```
  <PropertyGroup>
```

```
    <PublishProtocol>FileSystem</PublishProtocol>
```

```
    <Configuration>Release</Configuration>
```

```
<Platform>Any CPU</Platform>

<TargetFramework>net462</TargetFramework>

<PublishDir>c:\temp\jiraetl</PublishDir>

</PropertyGroup>

</Project>
```

ApplicationEventConsumer.cs (Monitoring.ETL.Domain\RabbitMQ\ApplicationEvent\ApplicationEventConsumer.cs):

```
//using Monitoring.ETL.Process;

//namespace Monitoring.ETL.Domain.RabbitMQ.ApplicationEvent
//{

// public class ApplicationEventConsumer : RabbitMQConsumer<ApplicationEventModel>

// {

//     public ApplicationEventConsumer(IEtlProcessLogger logger)

//         : base(new ApplicationEventRabbitMQConfiguration(), logger)

//     {

//     }

// }

//}
```

ApplicationEventExtractor.cs (Monitoring.ETL.Domain\RabbitMQ\ApplicationEvent\ApplicationEventExtractor.cs):

```
//using System;

//using System.Collections.Generic;

//using System.Threading;

//using System.Threading.Tasks;

//using ETL.Process;
```

```

//using Monitoring.ETL.Process;

//namespace Monitoring.ETL.Domain.RabbitMQ.ApplicationEvent

//{{

// public class ApplicationEventExtractor : IExtractor<ApplicationEventModel>

// {

//     private ApplicationEventConsumer _eventRepository;

//     public async Task<ResultSet<ApplicationEventModel>> ExtractAllSinceLastExtractAsync(
//         CancellationToken cancellationToken, IEtlProcessLogger logger)
//     {

//         _eventRepository = _eventRepository ?? new ApplicationEventConsumer(logger);

//         var events = await _eventRepository.ExtractAsync();

//         return events;

//     }

//     public Task<IEnumerable<ApplicationEventModel>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
//         CancellationToken cancellationToken, IEtlProcessLogger logger)
//     {

//         _eventRepository = _eventRepository ?? new ApplicationEventConsumer(logger);

//         throw new NotImplementedException();

//     }

// }

//}

```

ApplicationEventProducer.cs (Monitoring.ETL.Domain\RabbitMQ\ApplicationEvent\ApplicationEventProducer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.ApplicationEvent

{

    public sealed class ApplicationEventProducer : RabbitMQProducer<RabbitMQ.Model.ApplicationEvent>

    {

        public ApplicationEventProducer()

            : base(new Configuration())

        {

        }

    }

}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\ApplicationEvent\Configuration.cs):

```
using System.Configuration;

namespace Monitoring.ETL.Domain.RabbitMQ.ApplicationEvent

{

    public sealed class Configuration : MonitoringEventsConfiguration

    {

        public Configuration()

        {

            Queue = "monitoring.warehouse.applicationevents";

            RoutingKeyMessageSourceComponent = "applicationevent";

            BatchSize = EtlSettings.BatchSize(5000);

        }

    }

}
```

```
}  
  
}
```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\ApplicationEvent\Consumer.cs):

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.ApplicationEvent  
{  
  
    public class Consumer : RabbitMQConsumer<RabbitMQ.Model.ApplicationEvent>  
    {  
  
        public Consumer(IEtlProcessLogger logger)  
            : base(new Configuration(), logger)  
        {  
  
        }  
  
    }  
  
}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\Build\Configuration.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.Build  
{  
  
    public class Configuration : MonitoringDimensionsConfiguration  
    {  
  
        public Configuration()  
        {  
  
            Queue = "monitoring.warehouse.builds";  
  
            RoutingKeyMessageSourceComponent = "build";  
  
        }  
  
    }  
  
}
```

```
}  
  
}  
  
}
```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\Build\Consumer.cs):

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Build
```

```
{  
  
    public class Consumer : RabbitMQConsumer<RabbitMQ.Model.Build>  
  
    {  
  
        public Consumer(IEtlProcessLogger logger)  
  
            : base(new Configuration(), logger)  
  
        {  
  
        }  
  
    }  
  
}
```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\Build\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.Build
```

```
{  
  
    public class Producer : RabbitMQProducer<RabbitMQ.Model.Build>  
  
    {  
  
        public Producer()  
  
            : base(new Configuration())  
  
        {  
  
        }  
  
    }  
  
}
```



```
}  
  
}  
  
}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\ClassicDataSource\Configuration.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.ClassicDataSource  
  
{  
  
    public class Configuration : MonitoringDimensionsConfiguration  
  
    {  
  
        public Configuration()  
  
        {  
  
            Queue = "monitoring.warehouse.classicdatasources";  
  
            RoutingKeyMessageSourceComponent = "classicdatasource";  
  
        }  
  
    }  
  
}
```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\ClassicDataSource\Consumer.cs):

```
using Monitoring.ETL.Process;  
  
namespace Monitoring.ETL.Domain.RabbitMQ.ClassicDataSource  
  
{  
  
    public class Consumer : RabbitMQConsumer<Model.RabbitMQClassicDataSource>  
  
    {  
  
        public Consumer(IEtlProcessLogger logger)  
  
            : base(new Configuration(), logger)
```

```
{  
  
}  
  
}  
  
}
```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\ClassicDataSource\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.ClassicDataSource  
  
{  
  
    public class Producer : RabbitMQProducer<Model.RabbitMQClassicDataSource>  
  
    {  
  
        public Producer()  
  
            : base(new Configuration())  
  
        {  
  
        }  
  
    }  
  
}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\ClassicScreen\Configuration.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.ClassicScreen  
  
{  
  
    public class Configuration : MonitoringDimensionsConfiguration  
  
    {  
  
        public Configuration()  
  
        {  
  
            Queue = "monitoring.warehouse.classicscreens";  
  
            RoutingKeyMessageSourceComponent = "classicscreen";  
  
        }  
  
    }  
  
}
```

```
}  
  
}  
  
}
```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\ClassicScreen\Consumer.cs):

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.ClassicScreen
```

```
{  
  
    public class Consumer : RabbitMQConsumer<Model.RabbitMQClassicScreen>  
  
    {  
  
        public Consumer(IEtlProcessLogger logger)  
  
            : base(new Configuration(), logger)  
  
        {  
  
        }  
  
    }  
  
}
```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\ClassicScreen\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.ClassicScreen
```

```
{  
  
    public class Producer : RabbitMQProducer<Model.RabbitMQClassicScreen>  
  
    {  
  
        public Producer()  
  
            : base(new Configuration())  
  
        {  
  
        }  
  
    }  
  
}
```

```
}  
  
}  
  
}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\CloudDataSource\Configuration.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.CloudDataSource  
  
{  
  
    public class Configuration : MonitoringDimensionsConfiguration  
  
    {  
  
        public Configuration()  
  
        {  
  
            Queue = "monitoring.warehouse.clouddatasources";  
  
            RoutingKeyMessageSourceComponent = "clouddatasource";  
  
        }  
  
    }  
  
}
```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\CloudDataSource\Consumer.cs):

```
using Monitoring.ETL.Process;  
  
namespace Monitoring.ETL.Domain.RabbitMQ.CloudDataSource  
  
{  
  
    public class Consumer : RabbitMQConsumer<Model.RabbitMQCloudDataSource>  
  
    {  
  
        public Consumer(IEtlProcessLogger logger)  
  
            : base(new Configuration(), logger)
```

```
{  
  
}  
  
}  
  
}
```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\CloudDataSource\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.CloudDataSource  
  
{  
  
    public class Producer : RabbitMQProducer<Model.RabbitMQCloudDataSource>  
  
    {  
  
        public Producer()  
  
            : base(new Configuration())  
  
        {  
  
        }  
  
    }  
  
}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\Customer\Configuration.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.Customer  
  
{  
  
    public class Configuration : MonitoringDimensionsConfiguration  
  
    {  
  
        public Configuration()  
  
        {  
  
            Queue = "monitoring.warehouse.customers";  
  
            RoutingKeyMessageSourceComponent = "customer";  
  
        }  
  
    }  
  
}
```

```
}  
  
}  
  
}
```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\Customer\Consumer.cs):

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Customer
```

```
{  
  
    public class Consumer : RabbitMQConsumer<RabbitMQ.Model.Customer>  
  
    {  
  
        public Consumer(IEtlProcessLogger logger)  
  
            : base(new Configuration(), logger)  
  
        {  
  
        }  
  
    }  
  
}
```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\Customer\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.Customer
```

```
{  
  
    public class Producer : RabbitMQProducer<RabbitMQ.Model.Customer>  
  
    {  
  
        public Producer()  
  
            : base(new Configuration())  
  
        {  
  
        }  
  
    }  
  
}
```

```
}  
  
}  
  
}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\DatabasePerformanceMeasurement\Configuration.cs):

```
using System.Configuration;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.DatabasePerformanceMeasurement  
{  
  
    public class Configuration : MonitoringEventsConfiguration  
  
    {  
  
        public Configuration()  
  
        {  
  
            Queue = "monitoring.warehouse.databasemeasurements";  
  
            RoutingKeyMessageSourceComponent = "databasemeasurement";  
  
            BatchSize = EtlSettings.BatchSize(5);  
  
        }  
  
    }  
  
}
```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\DatabasePerformanceMeasurement\Consumer.cs):

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.DatabasePerformanceMeasurement  
{  
  
    public class Consumer : RabbitMQConsumer<RabbitMQ.Model.DatabasePerformanceMeasurement>
```

```

{

    public Consumer(IEtlProcessLogger logger) : base(new Configuration(), logger)

    {

    }

}
}

```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\DatabasePerformanceMeasurement\Producer.cs):

```

namespace Monitoring.ETL.Domain.RabbitMQ.DatabasePerformanceMeasurement

{

    public class Producer : RabbitMQProducer<RabbitMQ.Model.DatabasePerformanceMeasurement>

    {

        public Producer() : base(new Configuration())

        {

        }

    }

}

```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\DataSource\Configuration.cs):

```

using System.Configuration;

namespace Monitoring.ETL.Domain.RabbitMQ.DataSource

{

    public class Configuration : MonitoringDimensionsConfiguration

    {

        public Configuration()

    }

}

```



```
{  
  
    Queue = "monitoring.warehouse.webservicetransactions";  
  
    RoutingKeyMessageSourceComponent = "webservicetransaction";  
  
}  
  
}  
  
}
```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\DataSource\Consumer.cs):

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.DataSource  
{  
  
    public class Consumer : RabbitMQConsumer<RabbitMQ.Model.DataSource>  
  
    {  
  
        public Consumer(IEtlProcessLogger logger)  
  
            : base(new Configuration(), logger)  
  
        {  
  
        }  
  
    }  
  
}
```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\DataSource\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.DataSource  
{  
  
    public sealed class Producer : RabbitMQProducer<RabbitMQ.Model.DataSource>  
  
    {
```

```

public Producer()

    : base(new Configuration())

{

}

}

}

```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\Deployment\Configuration.cs):

```

namespace Monitoring.ETL.Domain.RabbitMQ.Deployment

{

    internal class Configuration : MonitoringDimensionsConfiguration

    {

        public Configuration()

        {

            Queue = "monitoring.warehouse.deployments";

            RoutingKeyMessageSourceComponent = "deployment";

        }

    }

}

```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\Deployment\Consumer.cs):

```

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.RabbitMQ.Deployment

{

    internal class Consumer : RabbitMQConsumer<RabbitMQ.Model.Deployment>

```

```

{

    public Consumer(IEtlProcessLogger logger)

        : base(new Configuration(), logger)

    {

    }

}

}

```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\Deployment\Producer.cs):

```

namespace Monitoring.ETL.Domain.RabbitMQ.Deployment

{

    internal class Producer : RabbitMQProducer<RabbitMQ.Model.Deployment>

    {

        public Producer()

            : base(new Configuration())

        {

        }

    }

}

```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\EDIUsage\Configuration.cs):

```

namespace Monitoring.ETL.Domain.RabbitMQ.EDIUsage

{

    public class Configuration : MonitoringEventsConfiguration

    {

        public Configuration()

    }

}

```

```

{

    Queue = "monitoring.warehouse.edi.usage";

    RoutingKeyMessageSourceComponent = "ediusage"; //this is a friendly name to identify this source of data

}

}

}

```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\EDIUsage\Consumer.cs):

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.EDIUsage
```

```

{

    public class Consumer : RabbitMQConsumer<Model.RabbitMQEDIUsageModel>

    {

        /// <summary>

        /// Creates a new logger for you to log information to.

        /// </summary>

        /// <param name="logger"></param>

        public Consumer(IEtlProcessLogger logger) : base(new Configuration(), logger)

        {

        }

    }

}

```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\EDIUsage\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.EDIUsage
```

```

{

    public class Producer : RabbitMQProducer<Model.RabbitMQEDIUsageModel>

    {

        public Producer() : base(new Configuration())

        {

        }

    }

}

```

IJsonLoadModel.cs (Monitoring.ETL.Domain\RabbitMQ\IJsonLoadModel.cs):

```

namespace Monitoring.ETL.Domain.RabbitMQ

{

    public interface IJsonLoadModel

    {

        string JsonMessage { get; set; }

    }

}

```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\ImpactAnalysis\Configuration.cs):

```

using System.Configuration;

namespace Monitoring.ETL.Domain.RabbitMQ.ImpactAnalysis

{

    public class Configuration : MonitoringEventsConfiguration

    {

        public Configuration()

    }

}

```

```

{
    Queue = "monitoring.warehouse.impactanalysis";

    RoutingKeyMessageSourceComponent = "impactanalysis";

    BatchSize = EtlSettings.BatchSize(5000);
}
}
}

```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\ImpactAnalysis\Consumer.cs):

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.ImpactAnalysis
```

```

{
    public class Consumer : RabbitMQConsumer<RabbitMQ.Model.ImpactAnalysis>
    {
        public Consumer(IEtlProcessLogger logger)
            : base(new Configuration(), logger)
        {
        }
    }
}

```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\ImpactAnalysis\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.ImpactAnalysis
```

```
{
```

```

public class Producer : RabbitMQProducer<RabbitMQ.Model.ImpactAnalysis>
{
    public Producer()
        : base(new Configuration())
    {

    }
}

```

IRabbitMQConfiguration.cs (Monitoring.ETL.Domain\RabbitMQ\IRabbitMQConfiguration.cs):

```

namespace Monitoring.ETL.Domain.RabbitMQ
{
    public interface IRabbitMQConfiguration
    {
        ushort BatchSize { get; }

        string VirtualHost { get; }

        string Queue { get; }

        string Exchange { get; }

        string ExchangeType { get; }

        string RoutingKeyMessageSourceComponent { get; }

        bool DiscardMalformedMessages { get; }
    }
}

```

IRabbitMQExtractModel.cs (Monitoring.ETL.Domain\RabbitMQ\IRabbitMQExtractModel.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ
```

```
{
```

```
    public interface IRabbitMQExtractModel : IExtractModel
```

```
    {
```

```
    }
```

```
}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\Jira\Configuration.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.Jira
```

```
{
```

```
    public class Configuration : MonitoringEventsConfiguration
```

```
    {
```

```
        public Configuration()
```

```
        {
```

```
            Queue = "monitoring.warehouse.jiraissues";
```

```
            RoutingKeyMessageSourceComponent = "jiraissue";
```

```
            BatchSize = 50;
```

```
        }
```

```
    }
```

```
}
```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\Jira\Consumer.cs):

```
using Monitoring.ETL.Process;
```



```
namespace Monitoring.ETL.Domain.RabbitMQ.Jira
```

```
{  
  
    public class Consumer : RabbitMQConsumer<Model.RabbitMQJiraIssue>  
  
    {  
  
        public Consumer(IEtlProcessLogger logger) : base(new Configuration(), logger)  
  
        {  
  
        }  
  
    }  
  
}
```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\Jira\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.Jira  
  
{  
  
    public class Producer : RabbitMQProducer<Model.RabbitMQJiraIssue>  
  
    {  
  
        public Producer() : base(new Configuration())  
  
        {  
  
        }  
  
    }  
  
}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\Kpis\Configuration.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.Kpis  
  
{  
  
    public class Configuration : MonitoringDimensionsConfiguration  
  
    {
```

```

public Configuration()

{

    Queue = "monitoring.warehouse.kpis";

    RoutingKeyMessageSourceComponent = "kpis";

}

}

}

```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\Kpis\Consumer.cs):

```

using Monitoring.ETL.Domain.Kpis.Source.Models;

using Monitoring.ETL.Domain.RabbitMQ.Model;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.RabbitMQ.Kpis

{

    public class Consumer : RabbitMQConsumer<RabbitMQKpiDataPoint>

    {

        public Consumer(IEtlProcessLogger logger)

            : base(new Configuration(), logger)

        {

        }

    }

}

```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\Kpis\Producer.cs):

```

using Monitoring.ETL.Domain.Kpis.Source.Models;

```

```
using Monitoring.ETL.Domain.RabbitMQ.Model;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Kpis
```

```
{  
  
    public class Producer : RabbitMQProducer<RabbitMQKpiDataPoint>  
  
    {  
  
        public Producer()  
  
            : base(new Configuration())  
  
        {  
  
        }  
  
    }  
}
```

Loader.cs (Monitoring.ETL.Domain\RabbitMQ\Loader.cs):

```
using System.Collections.Generic;  
  
using System.Threading;  
  
using System.Threading.Tasks;  
  
using ETL.Process;  
  
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ
```

```
{  
  
    public abstract class Loader<T> : ILoader<T>  
  
        where T : ILoadModel  
  
    {  
  
        private readonly RabbitMQProducer<T> _producer;
```

```

internal protected Loader(RabbitMQProducer<T> producer)

{
    _producer = producer;
}

public async Task<bool> LoadAsync(IEnumerable<T> transformed, CancellationToken cancellationToken,
IEtlProcessLogger logger)

{
    await Task.Run(
        () =>
        {
            foreach (var model in transformed)
            {
                _producer.Publish(model);
            }
        });

    return true;
}
}
}

```

LoginConsumer.cs (Monitoring.ETL.Domain\RabbitMQ>Login>LoginConsumer.cs):

```
using Monitoring.ETL.Domain.Login.Load;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Login
```

```
{  
  
    public sealed class LoginConsumer : RabbitMQConsumer<FactLoginWarehouseLoadModel>  
  
    {  
  
        public LoginConsumer(IEtlProcessLogger logger)  
  
            : base(new LoginRabbitMQConfiguration(), logger)  
  
        {  
  
        }  
  
    }  
  
}
```

LoginProducer.cs (Monitoring.ETL.Domain\RabbitMQ\Login\LoginProducer.cs):

```
using Monitoring.ETL.Domain.Login.Load;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Login
```

```
{  
  
    public sealed class LoginProducer : RabbitMQProducer<FactLoginLoadModel>  
  
    {  
  
        public LoginProducer() : base(new LoginRabbitMQConfiguration())  
  
        {  
  
        }  
  
    }  
  
}
```

LoginRabbitMQConfiguration.cs (Monitoring.ETL.Domain\RabbitMQ\Login\LoginRabbitMQConfiguration.cs):

```
using System.Configuration;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Login
```

```
{
```

```
    public sealed class LoginRabbitMQConfiguration : IRabbitMQConfiguration
```

```
    {
```

```
        public LoginRabbitMQConfiguration()
```

```
        {
```

```
            var success = bool.TryParse(ConfigurationManager.AppSettings["RabbitMQ.DiscardMalformedMessages"],
```

```
                out var discardMalformedMessages);
```

```
            DiscardMalformedMessages = success && discardMalformedMessages;
```

```
            BatchSize = EtlSettings.BatchSize(1000);
```

```
        }
```

```
        public ushort BatchSize { get; set; }
```

```
        public string VirtualHost => "monitoring";
```

```
        public string Queue => "monitoring.warehouse.logins";
```

```
        public string Exchange => "monitoring.events";
```

```
        public string ExchangeType => "topic";
```

```
        public string RoutingKeyMessageSourceComponent => "login";
```

```
        public bool DiscardMalformedMessages { get; }
```

```
    }
```

```
}
```

ApplicationEvent.cs (Monitoring.ETL.Domain\RabbitMQ\Model\ApplicationEvent.cs):

using System;

using Monitoring.ETL.Domain.ElasticSearch;

using Monitoring.ETL.Domain.RabbitMQ;

using Newtonsoft.Json;

namespace Monitoring.ETL.Domain.RabbitMQ.Model

{

public class ApplicationEvent : IRabbitMQExtractModel

{

public string Elastic_Id { get; set; }

[JsonProperty("@timestamp")]

public DateTime Timestamp { get; set; }

public string Message { get; set; }

public string Event_Division { get; set; }

public string Event_Source { get; set; }

public string Event_Queue { get; set; }

public string Severity { get; set; }

public string Environment { get; set; }

[JsonProperty("@version")]

public string Version { get; set; }

public string Event_Type { get; set; }

[JsonConverter(typeof(StringArrayConverter))]

public string[] Event_Sub_Type { get; set; }

public string[] Event_Sub_Type_Matches { get; set; }

public int PUN { get; set; }

```
public int PCN { get; set; }

public string Destination_Hostname { get; set; }

public string Referer { get; set; }

public string Source_IP { get; set; }

public string Source_Hostname { get; set; }

public string Method { get; set; }

public string User_Agent { get; set; }

public string User_Agent_Device { get; set; }

public string User_Agent_Major { get; set; }

public string User_Agent_Name { get; set; }

public string User_Agent_OS { get; set; }

public string User_Agent_Patch { get; set; }

public string Data_Center { get; set; }

[JsonConverter(typeof(StringArrayConverter))]

public string[] Tags { get; set; }

public int Error_Key { get; set; }

public string Filename { get; set; }

public string Path_Info { get; set; }

public string Script_Name { get; set; }

public string Lock_Chain { get; set; }

public string Sql_Server { get; set; }

public string Path_Translated { get; set; }

public string Element_List { get; set; }

public string Server_Name { get; set; }

public string Session_Id { get; set; }

public string Session_Info { get; set; }
```



```

public string Setting_Group { get; set; }

public string Setting_Name { get; set; }

public string Exception_Data_Source_Key { get; set; }

public string Exception_Data_Source_Name { get; set; }

public string Exception_Column_Name { get; set; }

public string Exception_SQL_Command { get; set; }

public string Exception_Error_Message { get; set; }

public string Missing_Image_Path { get; set; }

public string Stack_Trace { get; set; }

public string Exception_Location { get; set; }

public string ApplicationName { get; set; }

public string Referer_Path { get; set; }

public string NodeId { get; set; }

public int? ThreadId { get; set; }

public long? PreviousChangeVersion { get; set; }

}

}

```

Build.cs (Monitoring.ETL.Domain\RabbitMQ\Model\Build.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model
```

```

{

    public class Build : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel, IRabbitMQExtractModel

    {

        public string JsonMessage { get; set; }

    }

}

```

```
}
```

```
}
```

Customer.cs (Monitoring.ETL.Domain\RabbitMQ\Model\Customer.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model
```

```
{
```

```
    public class Customer : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel, IRabbitMQExtractModel
```

```
    {
```

```
        public string JsonMessage { get; set; }
```

```
    }
```

```
}
```

DatabasePerformanceMeasurement.cs

(Monitoring.ETL.Domain\RabbitMQ\Model\DatabasePerformanceMeasurement.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model
```

```
{
```

```
    public class DatabasePerformanceMeasurement : Warehouse.Model.Performance_Analysis_Data_w,
```

```
        IRabbitMQExtractModel
```

```
    {
```

```
    }
```

```
}
```

DataSource.cs (Monitoring.ETL.Domain\RabbitMQ\Model\DataSource.cs):

using ETL.Process;

namespace Monitoring.ETL.Domain.RabbitMQ.Model

{

public class DataSource : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel, IRabbitMQExtractModel

{

public string JsonMessage { get; set; }

}

}

Deployment.cs (Monitoring.ETL.Domain\RabbitMQ\Model\Deployment.cs):

namespace Monitoring.ETL.Domain.RabbitMQ.Model

{

public class Deployment : Warehouse.Model.Deployment_w, IRabbitMQExtractModel

{

}

}

ImpactAnalysis.cs (Monitoring.ETL.Domain\RabbitMQ\Model\ImpactAnalysis.cs):

using ETL.Process;

namespace Monitoring.ETL.Domain.RabbitMQ.Model

{

public class ImpactAnalysis : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel, IRabbitMQExtractModel

{

```
        public string JsonMessage { get; set; }

    }

}
```

OdbcStatement.cs (Monitoring.ETL.Domain\RabbitMQ\Model\OdbcStatement.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model

{

    public class OdbcStatement : Warehouse.Model.ODBC_Statement_w, IRabbitMQExtractModel

    {

    }

}
```

Procedure.cs (Monitoring.ETL.Domain\RabbitMQ\Model\Procedure.cs):

```
using ETL.Process;

namespace Monitoring.ETL.Domain.RabbitMQ.Model

{

    public class Procedure : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel, IRabbitMQExtractModel

    {

        public string JsonMessage { get; set; }

    }

}
```

RabbitMQClassicDataSource.cs (Monitoring.ETL.Domain\RabbitMQ\Model\RabbitMQClassicDataSource.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model
```

```
{
```

```
    public class RabbitMQClassicDataSource : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel,  
    IRabbitMQExtractModel
```

```
    {
```

```
        public string JsonMessage { get; set; }
```

```
    }
```

```
}
```

RabbitMQClassicScreen.cs (Monitoring.ETL.Domain\RabbitMQ\Model\RabbitMQClassicScreen.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model
```

```
{
```

```
    public class RabbitMQClassicScreen : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel,  
    IRabbitMQExtractModel
```

```
    {
```

```
        public string JsonMessage { get; set; }
```

```
    }
```

```
}
```

RabbitMQCloudDataSource.cs (Monitoring.ETL.Domain\RabbitMQ\Model\RabbitMQCloudDataSource.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model
```

```
{
```

```

public class RabbitMQCloudDataSource : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel,
IRabbitMQExtractModel

{

    public string JsonMessage { get; set; }

}

}

```

RabbitMQEDIUsageModel.cs (Monitoring.ETL.Domain\RabbitMQ\Model\RabbitMQEDIUsageModel.cs):

```

using ETL.Process;

```

```

using Monitoring.ETL.Domain.SqlJson;

```

```

namespace Monitoring.ETL.Domain.RabbitMQ.Model

```

```

{

    public class RabbitMQEDIUsageModel : IExtractModel, ILoadModel, IJsonExtractModel, IRabbitMQExtractModel

    {

        public string JsonMessage { get; set; }

    }

}

```

RabbitMQJiraIssue.cs (Monitoring.ETL.Domain\RabbitMQ\Model\RabbitMQJiraIssue.cs):

```

using ETL.Process;

```

```

namespace Monitoring.ETL.Domain.RabbitMQ.Model

```

```

{

    /// <summary>

```

/// The model which gets transferred to Rabbit MQ for the Jira Issues.

/// </summary>

```
public class RabbitMQJiraIssue : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel, IRabbitMQExtractModel
{
    public string JsonMessage { get; set; }
}
}
```

RabbitMQKpiDataPoint.cs (Monitoring.ETL.Domain\RabbitMQ\Model\RabbitMQKpiDataPoint.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model
```

```
{
    public class RabbitMQKpiDataPoint : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel, IRabbitMQExtractModel
    {
        public string JsonMessage { get; set; }
    }
}
```

RabbitMQModule.cs (Monitoring.ETL.Domain\RabbitMQ\Model\RabbitMQModule.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model
```

```
{
    public class RabbitMQModule : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel, IRabbitMQExtractModel
    {
```

```
        public string JsonMessage { get; set; }

    }

}
```

RabbitMQTemplateModel.cs (Monitoring.ETL.Domain\RabbitMQ\Model\RabbitMQTemplateModel.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.SqlJson;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model
```

```
{

    public class RabbitMQTemplateModel : IExtractModel, ILoadModel, IJsonExtractModel, IRabbitMQExtractModel

    {

        public string JsonMessage { get; set; }

    }

}
```

Release.cs (Monitoring.ETL.Domain\RabbitMQ\Model\Release.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model
```

```
{

    public class Release : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel, IRabbitMQExtractModel

    {

        public string JsonMessage { get; set; }

    }

}
```



```
}
```

RockwellMfgIndex.cs (Monitoring.ETL.Domain\RabbitMQ\Model\RockwellMfgIndex.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model
```

```
{
```

```
    public class RockwellMfgIndex : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel, IRabbitMQExtractModel
```

```
    {
```

```
        public string JsonMessage { get; set; }
```

```
    }
```

```
}
```

SqlExecution.cs (Monitoring.ETL.Domain\RabbitMQ\Model\SqlExecution.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model
```

```
{
```

```
    public class SqlExecution : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel, IRabbitMQExtractModel
```

```
    {
```

```
        public string JsonMessage { get; set; }
```

```
    }
```

```
}
```

User.cs (Monitoring.ETL.Domain\RabbitMQ\Model\User.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model
```

```
{  
  
    public class User : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel, IRabbitMQExtractModel  
  
    {  
  
        public string JsonMessage { get; set; }  
  
    }  
  
}
```

WebServiceTransaction.cs (Monitoring.ETL.Domain\RabbitMQ\Model\WebServiceTransaction.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Model
```

```
{  
  
    public class WebServiceTransaction : IExtractModel, ILoadModel, SqlJson.IJsonExtractModel, IRabbitMQExtractModel  
  
    {  
  
        public string JsonMessage { get; set; }  
  
    }  
  
}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\Module\Configuration.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.Module
```

```
{  
  
    public class Configuration : MonitoringDimensionsConfiguration  
  
    {  
  
        public Configuration()  
  
    }  
  
}
```

```

{
    Queue = "monitoring.warehouse.modules";

    RoutingKeyMessageSourceComponent = "module";
}
}
}

```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQModule\Consumer.cs):

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Module
```

```

{
    public class Consumer : RabbitMQConsumer<Model.RabbitMQModule>
    {
        public Consumer(IEtlProcessLogger logger) : base(new Configuration(), logger)
        {
        }
    }
}

```

Producer.cs (Monitoring.ETL.Domain\RabbitMQModule\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.Module
```

```

{
    public class Producer : RabbitMQProducer<Model.RabbitMQModule>
    {
        public Producer() : base(new Configuration())
    }
}

```

```
{  
  
}  
  
}  
  
}
```

MonitoringDimensionsConfiguration.cs (Monitoring.ETL.Domain\RabbitMQ\MonitoringDimensionsConfiguration.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ  
  
{  
  
    public abstract class MonitoringDimensionsConfiguration : IRabbitMQConfiguration  
  
    {  
  
        public MonitoringDimensionsConfiguration()  
  
        {  
  
            var success = bool.TryParse(  
  
                System.Configuration.ConfigurationManager.AppSettings["RabbitMQ.DiscardMalformedMessages"],  
  
                out var discardMalformedMessages);  
  
  
            DiscardMalformedMessages = success && discardMalformedMessages;  
  
        }  
  
  
  
  
        public ushort BatchSize { get; protected set; } = 20000;  
  
        public string VirtualHost => "monitoring";  
  
        public string Queue { get; protected set; }  
  
        public string Exchange => "monitoring.dimensions";  
  
        public string ExchangeType => "topic";  
  
        public string RoutingKeyMessageSourceComponent { get; protected set; }  
  
        public bool DiscardMalformedMessages { get; private set; }  
  
    }  
  
}
```

```
}
```

```
}
```

MonitoringEventsConfiguration.cs (Monitoring.ETL.Domain\RabbitMQ\MonitoringEventsConfiguration.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Text;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ
```

```
{
```

```
    public abstract class MonitoringEventsConfiguration : IRabbitMQConfiguration
```

```
    {
```

```
        protected MonitoringEventsConfiguration()
```

```
        {
```

```
            var success = bool.TryParse(
```

```
                System.Configuration.ConfigurationManager.AppSettings["RabbitMQ.DiscardMalformedMessages"],
```

```
                out var discardMalformedMessages);
```

```
            DiscardMalformedMessages = success && discardMalformedMessages;
```

```
        }
```

```
        /// <summary>
```

```
        /// Default batch size of 20000
```

```
        /// </summary>
```

```

    public ushort BatchSize { get; protected set; } = 20000;

    public string VirtualHost => "monitoring";

    public string Queue { get; protected set; }

    public string Exchange => "monitoring.events";

    public string ExchangeType => "topic";

    public string RoutingKeyMessageSourceComponent { get; protected set; }

    public bool DiscardMalformedMessages { get; private set; }

}

}

```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\Procedure\Configuration.cs):

```

namespace Monitoring.ETL.Domain.RabbitMQ.Procedure

{

    public class Configuration : MonitoringDimensionsConfiguration

    {

        public Configuration()

        {

            Queue = "monitoring.warehouse.procedures";

            RoutingKeyMessageSourceComponent = "procedure";

        }

    }

}

```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\Procedure\Consumer.cs):

```

using Monitoring.ETL.Process;

```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Procedure
```

```
{  
  
    public class Consumer : RabbitMQConsumer<RabbitMQ.Model.Procedure>  
  
    {  
  
        public Consumer(IEtlProcessLogger logger)  
  
            : base(new Configuration(), logger)  
  
        {  
  
        }  
  
    }  
  
}
```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\Procedure\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.Procedure  
  
{  
  
    public class Producer : RabbitMQProducer<RabbitMQ.Model.Procedure>  
  
    {  
  
        public Producer()  
  
            : base(new Configuration())  
  
        {  
  
        }  
  
    }  
  
}
```

RabbitMQConnector.cs (Monitoring.ETL.Domain\RabbitMQ\RabbitMQConnector.cs):

```
using System;  
  
using System.Collections.Generic;
```

```
using System.Configuration;
```

```
using System.Globalization;
```

```
using Plex.Infrastructure.ConfigSystems;
```

```
using RabbitMQ.Client;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ
```

```
{
```

```
    public abstract class RabbitMQConnector
```

```
    {
```

```
        private readonly string _hostName;
```

```
        private readonly string _password;
```

```
        private readonly int _port;
```

```
        private readonly IPlexRegistrySystem _registrySystem = new PlexRegistrySystem();
```

```
        private readonly string _userName;
```

```
        private readonly TimeSpan _recoveryTimespan = new TimeSpan(0, 5, 0); //5 min wait on rabbit recovery
```

```
        protected RabbitMQConnector(IRabbitMQConfiguration configuration)
```

```
        {
```

```
            Configuration = configuration ?? throw new ArgumentNullException(nameof(configuration));
```

```
            _hostName = GetRabbitMqRegistryValueOrThrow("Host");
```

```
            _userName = GetRabbitMqRegistryValueOrThrow("Username");
```

```
            _password = GetRabbitMqRegistryValueOrThrow("Password");
```



```
var portString = GetRabbitMqRegistryValueOrThrow("Port");
```

```
if (!int.TryParse(portString, NumberStyles.Integer, CultureInfo.InvariantCulture, out _port))
```

```
{
```

```
    throw new ArgumentException("The port specified was not a valid integer. Value: " + portString);
```

```
}
```

```
ThrowIfConfigurationPropertyIsNullOrEmpty("VirtualHost", Configuration.VirtualHost);
```

```
}
```

```
protected IRabbitMQConfiguration Configuration { get; }
```

```
private ConnectionFactory GetConnectionFactory()
```

```
{
```

```
    return new ConnectionFactory
```

```
{
```

```
        HostName = _hostName,
```

```
        Port = _port,
```

```
        VirtualHost = Configuration.VirtualHost,
```

```
        UserName = _userName,
```

```
        Password = _password,
```

```
        DispatchConsumersAsync = true,
```

```
        AutomaticRecoveryEnabled = true,
```

```
        TopologyRecoveryEnabled = true,
```

```
        NetworkRecoveryInterval = _recoveryTimespan
```

```
};
```

```
}
```

```
protected IModel DeclareAndGetQueue()
```

```
{
```

```
    var connectionFactory = GetConnectionFactory();
```

```
    var connection = connectionFactory.CreateConnection();
```

```
    var channel = connection.CreateModel();
```

```
    channel.ExchangeDeclare(Configuration.Exchange, Configuration.ExchangeType, true, false);
```

```
    channel.QueueDeclare(Configuration.Queue, true, false, false);
```

```
    channel.QueueBind(Configuration.Queue, Configuration.Exchange, "#" +
```

```
Configuration.RoutingKeyMessageSourceComponent);
```

```
    return channel;
```

```
}
```

```
protected void AddOrUpdateRabbitMQInfoToException(Exception e)
```

```
{
```

```
    var keyValues = new Dictionary<object, object>{
```

```
        {"X-RabbitMQ-HostName", _hostName},
```

```
        {"X-RabbitMQ-Port", _port},
```

```
        {"X-RabbitMQ-VirtualHost", Configuration.VirtualHost},
```

```
        {"X-RabbitMQ-UserName", _userName},
```

```
        {"X-RabbitMQ-Queue", Configuration.Queue},
```

```
        {"X-RabbitMQ-Exchange", Configuration.Exchange},
```

```
        {"X-RabbitMQ-ExchangeType", Configuration.ExchangeType}
```

```
};
```

```
foreach (var keyValue in keyValues)
```

```
{
```

```
    if (e.Data.Contains(keyValue.Key))
```

```
    {
```

```
        e.Data[keyValue.Key] = keyValue.Value;
```

```
    }
```

```
    else
```

```
    {
```

```
        e.Data.Add(keyValue.Key, keyValue.Value);
```

```
    }
```

```
}
```

```
}
```

```
protected string GetRabbitMqRegistryValueOrThrow(string valueName)
```

```
{
```

```
    // Look up value name override in app config.
```

```
    var overrideValueName = ConfigurationManager.AppSettings[$"RabbitMQ.{valueName}"];
```

```
    if (!string.IsNullOrEmpty(overrideValueName))
```

```
    {
```

```
        valueName = overrideValueName;
```

```
    }
```

```
    if (!_registrySystem.TryGetString("CloudOps", "Monitoring", "RabbitMQ", valueName, out var value) ||
```

```

        string.IsNullOrEmpty(value))

    {

        throw new ArgumentException(

            $"Could not find {valueName} in the registry. Path: HKLM\\Software\\Plex\\CloudOps\\Monitoring\\RabbitMQ\\

Entry Name: {valueName}");

    }

    return value;

}

```

```

protected static void ThrowIfConfigurationPropertyIsNullOrEmpty(string propertyName, string propertyValue)

{

    if (string.IsNullOrEmpty(propertyValue))

    {

        throw new ArgumentException($"The {propertyName} on the configuration must not be empty.");

    }

}

}

}

```

RabbitMQConsumer.cs (Monitoring.ETL.Domain\RabbitMQ\RabbitMQConsumer.cs):

```

using System;

using System.Collections.Concurrent;

using System.Text;

using System.Threading.Tasks;

```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
using Newtonsoft.Json;
```

```
using RabbitMQ.Client;
```

```
using RabbitMQ.Client.Events;
```

```
using RabbitMQ.Client.Exceptions;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ
```

```
{
```

```
    public abstract class RabbitMQConsumer<T> : RabbitMQConnector
```

```
    {
        where T : IRabbitMQExtractModel
```

```
    {
```

```
        private IModel _channel;
```

```
        private readonly IEtlProcessLogger _logger;
```

```
        private ConcurrentQueue<BasicDeliverEventArgs> _messages;
```

```
        private readonly int _retryCount = 20;
```

```
        private readonly int _retrySeconds = 60;
```

```
        protected RabbitMQConsumer(IRabbitMQConfiguration configuration, IEtlProcessLogger logger)
```

```
        {
            : base(configuration)
```

```
        {
```

```
            var tryCount = 0;
```

```
            _logger = logger;
```

```
ThrowIfConfigurationPropertyIsNullOrEmpty("Queue", Configuration.Queue);
```

```
var channelInitialized = false;
```

```
try
```

```
{
```

```
    while (!channelInitialized && tryCount < _retryCount)
```

```
    {
```

```
        channelInitialized = InitializeChannel();
```

```
        tryCount++;
```

```
        if (!channelInitialized)
```

```
        {
```

```
            _logger.Information($"Retrying in {_retrySeconds} seconds");
```

```
            Task.Delay(TimeSpan.FromSeconds(_retrySeconds));
```

```
        }
```

```
    }
```

```
}
```

```
catch (Exception e)
```

```
{
```

```
    AddOrUpdateRabbitMQInfoToException(e);
```

```
    throw;
```

```
}
```

```
}
```

```
/// <summary>
```

```
/// Initialize a channel. Return T
```

```
/// </summary>
```

```
/// <returns></returns>
```

```
private bool InitializeChannel()
```

```
{
```

```
    _logger.Information("Initialize Channel...");
```

```
    try
```

```
    {
```

```
        _channel = DeclareAndGetQueue();
```

```
        _channel.BasicQos(0, (ushort)(Configuration.BatchSize * 2), false);
```

```
        _messages = _messages ?? new ConcurrentQueue<BasicDeliverEventArgs>();
```

```
        var consumer = new AsyncEventingBasicConsumer(_channel);
```

```
        consumer.Received += async (sender, message) =>
```

```
        {
```

```
            await Task.Yield();
```

```
            _messages.Enqueue(message);
```

```
        };
```

```
        _channel.BasicConsume(Configuration.Queue, false, consumer);
```

```
        return true; // we were successful at creating the channel
```

```
    }
```

```
    catch (Exception ex)
```

```
    {
```

```
        _logger.Information($"Exception trying to connect to RabbitMQ: {ex.GetType()} - {ex.Message}");
```

```
        return false;
```

```
}  
  
}
```

```
public async Task<ResultSet<T>> ExtractAsync()  
  
{  
  
    var results = new ResultSet<T>();  
  
    while (results.Results.Count < Configuration.BatchSize && _messages.Count > 0)  
  
    {  
  
        _messages.TryDequeue(out var message);  
  
        var body = Encoding.UTF8.GetString(message.Body);  
  
  
        try  
  
        {  
  
            var model = JsonConvert.DeserializeObject<T>(body);  
  
            _channel.BasicAck(message.DeliveryTag, false);  
  
            results.Results.Add(model);  
  
        }  
  
        catch (AlreadyClosedException)  
  
        {  
  
            _messages = new ConcurrentQueue<BasicDeliverEventArgs>();  
  
  
            while (true)  
  
            {  
  
                _logger.Information("Waiting for connection to be restored...");
```



```

        if (_channel?.IsClosed ?? true)
        {
            _logger.Information(_channel?.CloseReason?.ToString() ?? "Unknown close reason.");

            await Task.Delay(TimeSpan.FromSeconds(5));
        }
        else
        {
            break;
        }
    }
}

catch (Exception ex)
{
    var exception = new Exception("Error parsing json object", ex);

    _channel.BasicNack(message.DeliveryTag, false, !Configuration.DiscardMalformedMessages);

    exception.Data.Add("json", body);

    results.Exceptions.Add(exception);
}

return results;
}
}
}

```

RabbitMQProducer.cs (Monitoring.ETL.Domain\RabbitMQ\RabbitMQProducer.cs):

using System;

using System.Text;

using System.Text.RegularExpressions;

using Newtonsoft.Json;

using RabbitMQ.Client;

namespace Monitoring.ETL.Domain.RabbitMQ

{

public abstract class RabbitMQProducer<T> : RabbitMQConnector

{

private readonly IModel _channel;

private readonly string _routingKeySiteEnvironment;

protected RabbitMQProducer(IRabbitMQConfiguration configuration)

: base(configuration)

{

_routingKeySiteEnvironment = GetRabbitMqRegistryValueOrThrow("RoutingKeySiteEnvironment");

if (Regex.IsMatch(_routingKeySiteEnvironment, "^[a-z][a-z0-9]*\\. [a-z][a-z0-9]*\$") == false)

{

throw new ArgumentException(

\$"The Routing Key Site Environment component was not in the expected format (<site>.<environment>).

Value: {_routingKeySiteEnvironment}");

```
}
```

```
if (Regex.IsMatch(Configuration.RoutingKeyMessageSourceComponent, "^[a-z]+" ) == false)
```

```
{
```

```
    throw new ArgumentException(
```

```
        "The RoutingKeyMessageSourceComponent on the configuration was not the correct format. " +
```

```
        " It must be non-empty and start with a alpha character and be lower case.");
```

```
}
```

```
ThrowIfConfigurationPropertyIsNullOrEmpty("Exchange", Configuration.Exchange);
```

```
try
```

```
{
```

```
    _channel = DeclareAndGetQueue();
```

```
}
```

```
catch (Exception e)
```

```
{
```

```
    AddOrUpdateRabbitMQInfoToException(e);
```

```
    throw;
```

```
}
```

```
}
```

```
private string RoutingKey =>
```

```
    $"{_routingKeySiteEnvironment}.{Configuration.RoutingKeyMessageSourceComponent}";
```

```
public void Publish(T messageContent)
```

```

{
    var messageContentJsonString = messageContent is IJsonLoadModel ?
        ((IJsonLoadModel)messageContent).JsonMessage :
        JsonConvert.SerializeObject(messageContent);

    var messageContentBytes = Encoding.UTF8.GetBytes(messageContentJsonString);

    _channel.BasicPublish(Configuration.Exchange, RoutingKey, true, null, messageContentBytes);
}
}
}

```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\Release\Configuration.cs):

```

namespace Monitoring.ETL.Domain.RabbitMQ.Release
{
    public class Configuration : MonitoringDimensionsConfiguration
    {
        public Configuration()
        {
            Queue = "monitoring.warehouse.releases";

            RoutingKeyMessageSourceComponent = "release";
        }
    }
}

```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\Release\Consumer.cs):

```

using Monitoring.ETL.Process;

```

```
namespace Monitoring.ETL.Domain.RabbitMQ.Release
```

```
{  
  
    public class Consumer : RabbitMQConsumer<RabbitMQ.Model.Release>  
  
    {  
  
        public Consumer(IEtlProcessLogger logger)  
  
            : base(new Configuration(), logger)  
  
        {  
  
        }  
  
    }  
  
}
```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\Release\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.Release  
  
{  
  
    public class Producer : RabbitMQProducer<RabbitMQ.Model.Release>  
  
    {  
  
        public Producer()  
  
            : base(new Configuration())  
  
        {  
  
        }  
  
    }  
  
}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\RockwellMfgIndex\Configuration.cs):

```
using System.Configuration;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.RockwellMfgIndex
```

```
{  
  
    public class Configuration : MonitoringEventsConfiguration  
  
    {  
  
        public Configuration()  
  
        {  
  
            Queue = "monitoring.warehouse.rockwellmfgindex";  
  
            RoutingKeyMessageSourceComponent = "rockwellmfgindex";  
  
            BatchSize = EtlSettings.BatchSize(5000);  
  
        }  
  
    }  
  
}
```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\RockwellMfgIndex\Consumer.cs):

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.RockwellMfgIndex
```

```
{  
  
    public class Consumer : RabbitMQConsumer<RabbitMQ.Model.RockwellMfgIndex>  
  
    {  
  
        public Consumer(IEtlProcessLogger logger)  
  
            : base(new Configuration(), logger)  
  
        {  
  
        }  
  
    }  
  
}
```

```
}
```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\RockwellMfgIndex\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.RockwellMfgIndex
```

```
{  
  
    public class Producer : RabbitMQProducer<RabbitMQ.Model.RockwellMfgIndex>  
  
    {  
  
        public Producer()  
  
            : base(new Configuration())  
  
        {  
  
        }  
  
    }  
  
}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\SqlExecution\Configuration.cs):

```
using System.Configuration;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.SqlExecution
```

```
{  
  
    public class Configuration : MonitoringEventsConfiguration  
  
    {  
  
        public Configuration()  
  
        {  
  
            Queue = "monitoring.warehouse.sql executions";  
  
            RoutingKeyMessageSourceComponent = "sql execution";  
  
        }  
  
    }  
  
}
```

```
        BatchSize = EtlSettings.BatchSize(500);  
  
    }  
  
}  
  
}
```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\SqlExecution\Consumer.cs):

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.SqlExecution  
{  
  
    public class Consumer : RabbitMQConsumer<RabbitMQ.Model.SqlExecution>  
    {  
  
        public Consumer(IEtlProcessLogger logger)  
            : base(new Configuration(), logger)  
        {  
  
        }  
  
    }  
  
}
```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\SqlExecution\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.SqlExecution  
{  
  
    public class Producer : RabbitMQProducer<RabbitMQ.Model.SqlExecution>  
    {  
  
        public Producer()  
        {  
        }  
    }  
}
```



```
        : base(new Configuration())
    {
    }
}
}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\Template\Configuration.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.Template
{
    public class Configuration : MonitoringEventsConfiguration
    {
        public Configuration()
        {
            Queue = "monitoring.warehouse.templatequeue";

            RoutingKeyMessageSourceComponent = "templatedata"; //this is a friendly name to identify this source of data
        }
    }
}
```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\Template\Consumer.cs):

```
using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.RabbitMQ.Template
{
    public class Consumer : RabbitMQConsumer<Model.RabbitMQTemplateModel>
    {
```

```
/// <summary>
```

```
/// Creates a new logger for you to log information to.
```

```
/// </summary>
```

```
/// <param name="logger"></param>
```

```
public Consumer(IEtlProcessLogger logger) : base(new Configuration(), logger)
```

```
{
```

```
}
```

```
}
```

```
}
```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\Template\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.Template
```

```
{
```

```
public class Producer : RabbitMQProducer<Model.RabbitMQTemplateModel>
```

```
{
```

```
public Producer() : base(new Configuration())
```

```
{
```

```
}
```

```
}
```

```
}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\User\Configuration.cs):

```
using System.Configuration;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.User
```

```
{
```

```

public class Configuration : IRabbitMQConfiguration

{

    public Configuration()

    {

        var success = bool.TryParse(ConfigurationManager.AppSettings["RabbitMQ.DiscardMalformedMessages"],

            out var discardMalformedMessages);

        DiscardMalformedMessages = success && discardMalformedMessages;

    }

    public ushort BatchSize { get; set; }

    public string VirtualHost => "monitoring";

    public string Queue => "monitoring.warehouse.users";

    public string Exchange => "monitoring.events";

    public string ExchangeType => "topic";

    public string RoutingKeyMessageSourceComponent => "users";

    public bool DiscardMalformedMessages { get; }

    }

}

```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\User\Consumer.cs):

```

using Monitoring.ETL.Process;

```

```

namespace Monitoring.ETL.Domain.RabbitMQ.User

```

```

{

```

```
public class Consumer : RabbitMQConsumer<RabbitMQ.Model.User>

{

    public Consumer(IEtlProcessLogger logger)

        : base(new Configuration(), logger)

    {

    }

}

}
```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\User\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.User

{

    public class Producer : RabbitMQProducer<RabbitMQ.Model.User>

    {

        public Producer() : base(new Configuration())

        {

        }

    }

}
```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\WebServices\Configuration.cs):

```
using System.Configuration;

namespace Monitoring.ETL.Domain.RabbitMQ.WebServices

{

    public class Configuration : MonitoringEventsConfiguration
```

```

{

    public Configuration()

    {

        Queue = "monitoring.warehouse.webservicetransactions";

        RoutingKeyMessageSourceComponent = "webservicetransaction";

        BatchSize = EtlSettings.BatchSize(5000);

    }

}
}

```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\WebServices\Consumer.cs):

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.WebServices
```

```

{

    public class Consumer : RabbitMQConsumer<Model.WebServiceTransaction>

    {

        public Consumer(IEtlProcessLogger logger)

            : base(new Configuration(), logger)

        {

        }

    }

}

```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\WebServices\Producer.cs):

```
namespace Monitoring.ETL.Domain.RabbitMQ.WebServices
```

```

{

public class Producer : RabbitMQProducer<Model.WebServiceTransaction>

{

    public Producer()

        : base(new Configuration())

    {

    }

}

}

```

Configuration.cs (Monitoring.ETL.Domain\RabbitMQ\WindowsServerFailoverCluster\Configuration.cs):

```

namespace Monitoring.ETL.Domain.RabbitMQ.WindowsServerFailoverCluster

{

    public sealed class Configuration : MonitoringEventsConfiguration

    {

        public Configuration()

        {

            BatchSize = EtlSettings.BatchSize(100);

            Queue = "monitoring.warehouse.windowsserverfailoverclusternoderolestates";

            RoutingKeyMessageSourceComponent = "windowsserverfailoverclusternoderolestate";

        }

    }

}

```

Consumer.cs (Monitoring.ETL.Domain\RabbitMQ\WindowsServerFailoverCluster\Consumer.cs):

```

using Monitoring.ETL.Domain.WindowsServerFailoverCluster;

```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.WindowsServerFailoverCluster
```

```
{
```

```
    public class Consumer : RabbitMQConsumer<ClusterNodeRoleModel>
```

```
    {
```

```
        public Consumer(IEtlProcessLogger logger)
```

```
            : base(new Configuration(), logger)
```

```
        {
```

```
        }
```

```
    }
```

```
}
```

Producer.cs (Monitoring.ETL.Domain\RabbitMQ\WindowsServerFailoverCluster\Producer.cs):

```
using Monitoring.ETL.Domain.WindowsServerFailoverCluster;
```

```
namespace Monitoring.ETL.Domain.RabbitMQ.WindowsServerFailoverCluster
```

```
{
```

```
    public sealed class Producer : RabbitMQProducer<ClusterNodeRoleModel>
```

```
    {
```

```
        public Producer() : base(new Configuration())
```

```
        {
```

```
        }
```

```
    }
```

```
}
```

DelayFactory.cs (Monitoring.ETL.Domain\RockwellMfgIndex\RMQExtractor\DelayFactory.cs):

using System;

namespace Monitoring.ETL.Domain.RockwellMfgIndex.RMQExtractor

```
{  
  
    public class DelayFactory : ThresholdExtractionDelayFactory<RabbitMQ.Model.RockwellMfgIndex>  
  
    {  
  
        protected override TimeSpan DelayTimeSpan => new TimeSpan(0, 5, 0);  
  
        protected override int MaxThresholdForDelay => 1000;  
  
    }  
}
```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\RockwellMfgIndex\RMQExtractor\RabbitMQExtractor.cs):

using System;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.RockwellMfgIndex.RMQExtractor

```
{  
  
    public sealed class RabbitMQExtractor : IExtractor<RabbitMQ.Model.RockwellMfgIndex>  
  
    {
```



```
private RabbitMQ.RockwellMfgIndex.Consumer _consumer;
```

```
public async Task<ResultSet<RabbitMQ.Model.RockwellMfgIndex>>
```

```
ExtractAllSinceLastExtractAsync(Cancellation_token cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    _consumer = _consumer ?? new RabbitMQ.RockwellMfgIndex.Consumer(logger);  
  
    return await _consumer.ExtractAsync();  
  
}
```

```
public Task<IEnumerable<RabbitMQ.Model.RockwellMfgIndex>> ExtractBetweenAsync(DateTime startDate,
```

```
DateTime endDate, Cancellation_token cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    throw new NotImplementedException();  
  
}  
  
}  
  
}
```

RabbitMQLoader.cs (Monitoring.ETL.Domain\RockwellMfgIndex\RMQExtractor\RabbitMQLoader.cs):

```
namespace Monitoring.ETL.Domain.RockwellMfgIndex.RMQExtractor
```

```
{  
  
    public class RabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.RockwellMfgIndex>  
  
    {  
  
        public RabbitMQLoader() : base(new RabbitMQ.RockwellMfgIndex.Producer())  
  
        {  
  
        }  
  
    }  
  
}
```

```
}
```

Transformer.cs (Monitoring.ETL.Domain\RockwellMfgIndex\RMQExtractor\Transformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RockwellMfgIndex.RMQExtractor
```

```
{
```

```
    /// <summary>
```

```
    /// Load the messages from RabbitMQ into the table Rockwell_Mfg_Index_w.
```

```
    /// </summary>
```

```
    public class Transformer : ITransformer<RabbitMQ.Model.RockwellMfgIndex,  
Warehouse.Model.Rockwell_Mfg_Index_w>
```

```
    {
```

```
        public async Task<IEnumerable<Warehouse.Model.Rockwell_Mfg_Index_w>>
```

```
TransformAsync(IEnumerable<RabbitMQ.Model.RockwellMfgIndex> extracted, CancellationToken cancellationToken,
```

```
IEtlProcessLogger logger)
```

```
    {
```

```
        await Task.Yield();
```

```

return extracted.Select(pc => new Warehouse.Model.Rockwell_Mfg_Index_w
{
    JSON_Message = pc.JsonMessage
});
}
}
}

```

WarehouseLoader.cs (Monitoring.ETL.Domain\RockwellMfgIndex\RMQExtractor\WarehouseLoader.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```
using Monitoring.ETL.Domain.Warehouse.Model;
```

```
namespace Monitoring.ETL.Domain.RockwellMfgIndex.RMQExtractor
```

```

{
    /// <summary>
    /// Execute the Fact_Rockwell_Mfg_Index_Add which takes the JSON from Rockwell_Mfg_Index_w and loads the
    Fact_Rockwell_Mfg_Index table.
    /// This sproc loads all messages, for Releases, DirectMaterialCost, InventoryDaysOnHand, MROCost,
    ProductionRecord and ProductionRate
    /// </summary>
    public class WarehouseLoader : Loader<Rockwell_Mfg_Index_w>
    {
        public WarehouseLoader() : base("Fact_Rockwell_Mfg_Index_Add", true)
        {
        }
    }
}

```

```
}
```

DelayFactory.cs (Monitoring.ETL.Domain\RockwellMfgIndex\RMQLoader\DelayFactory.cs):

```
using System;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RockwellMfgIndex.RMQLoader
```

```
{
```

```
    public class DelayFactory : IExtractionDelayFactory<RockwellMfgIndexModel>
```

```
    {
```

```
        private readonly TimeSpan _shortDelayTimespan = new TimeSpan(0, 10, 0);
```

```
        private readonly TimeSpan _longDelayTimespan = new TimeSpan(1, 0, 0);
```

```
        private readonly int _maxThresholdForDelay = 1;
```

```
        public async Task Create(ResultSet<RockwellMfgIndexModel> results, CancellationToken cancellationToken,
        IEtlProcessLogger logger)
```

```
        {
```

```
            if (results.Exceptions.Count > 0)
```

```
            {
```

```
                if (results.Exceptions.Any(e => (e is EmptyResultsException)))
```

```

{
    logger.Information("The number of results was zero for a historical date. Skipping delay.");
} else
{
    logger.Information($"Extraction incomplete due to exception(s). Retrying in
{_shortDelayTimespan.TotalSeconds} seconds.");

    await Task.Delay(_shortDelayTimespan, cancellationToken);
}
} else
{
    bool enoughResults = results != null && results.Results.Count >= _maxThresholdForDelay;

    if (enoughResults)
    {
        logger.Information($"The number of results ({results.Results.Count}) was beyond the max threshold of
{_maxThresholdForDelay}. Skipping delay.");
    } else
    {
        logger.Information($"Sleeping for {_longDelayTimespan.TotalSeconds} seconds.");

        await Task.Delay(_longDelayTimespan, cancellationToken);
    }
}
}
}
}

```

EmptyResultsException.cs (Monitoring.ETL.Domain\RockwellMfgIndex\RMQLoader\EmptyResultsException.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.RockwellMfgIndex.RMQLoader
```

```
{
```

```
    public class EmptyResultsException : Exception
```

```
    {
```

```
    }
```

```
}
```

Extractor.cs (Monitoring.ETL.Domain\RockwellMfgIndex\RMQLoader\Extractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using System.Configuration;
```

```
using System.Linq;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.ElasticSearch;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RockwellMfgIndex.RMQLoader
```

```
{
```

```
    public class Extractor : IExtractor<RockwellMfgIndexModel>
```

```
    {
```

```
private readonly ServiceToolConnectionRepository _serviceToolConnectionRepository;
```

```
private readonly List<Repository> _repositories;
```

```
private DateTime _minDate = new DateTime(2018, 1, 1);
```

```
private const string ServiceToolName = "Rockwell Mfg Index ETL";
```

```
public Extractor()
```

```
{
```

```
    if (DateTime.TryParse(ConfigurationManager.AppSettings["RMI_StartDate"], out DateTime startingDate))
```

```
    {
```

```
        if (DateTime.Now.Date > startingDate)
```

```
        {
```

```
            _minDate = startingDate;
```

```
        }
```

```
    }
```

```
    _serviceToolConnectionRepository = new ServiceToolConnectionRepository();
```

```
    _repositories = RepositoryLoader.Load();
```

```
}
```

```
public async Task<ResultSet<RockwellMfgIndexModel>> ExtractAllSinceLastExtractAsync(CancellationTokencancellationTokentoken, IEtlProcessLogger logger)
```

```
{
```

```
    var resultSet = new ResultSet<RockwellMfgIndexModel>();
```

```
    var loadStateRepository = new LoadStateRepository();
```

```
    try
```

```
{

    logger.Information("Retrieving Servers to process");

    var servers = _serviceToolConnectionRepository.GetServersToProcess(ServiceToolName).OrderBy(kvp =>
kvp.Key).ToList();

    logger.Information($"{servers.Count} server(s) found");

    var tasks = new List<Task<NScaleTransaction>>();

    var emptyResultsAdded = false;

    foreach (var server in servers)
    {
        tasks.Add(ExtractFromNScaleServer(server, cancellation_token, logger));
    }

    foreach (var transaction in await Task.WhenAll(tasks))
    {
        foreach (var result in transaction.Results)
        {
            resultSet.Results.Add(result);
        }

        foreach (var exception in transaction.Exceptions)
        {
            if (!(exception is EmptyResultsException) || !emptyResultsAdded)
```



```
{  
  
    resultSet.Exceptions.Add(exception);  
  
}
```

```
  
    emptyResultsAdded = exception is EmptyResultsException;  
  
}
```

```
  
foreach (var loadStateUpdate in transaction.LoadStateUpdates)  
  
{  
  
    await loadStateRepository.SetStateAsync(loadStateUpdate.Key, loadStateUpdate.Value);  
  
}  
  
}  
  
}  
  
catch (Exception e)  
  
{  
  
    resultSet.Exceptions.Add(e);  
  
}  
  
  
return resultSet;  
  
}
```

```
  
  
public Task<IEnumerable<RockwellMfgIndexModel>> ExtractBetweenAsync(DateTime startDate, DateTime  
endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)  
  
{  
  
    throw new NotImplementedException();  
  
}
```

```

private async Task<NScaleTransaction> ExtractFromNScaleServer(KeyValuePair<string,
ServiceToolConnectionRepository.SqlServerModel> server, CancellationToken cancellationToken, IEtlProcessLogger
logger)

{

    var serviceName = server.Key;

    var sqlServerModel = server.Value;

    var serverName = sqlServerModel.MachineName + (sqlServerModel.Port > 0 ? $",{sqlServerModel.Port}" :
$"\\{sqlServerModel.InstanceName}");

    var nScaleTransaction = new NScaleTransaction();

    var loadStateRepository = new LoadStateRepository();

    DateTime now = DateTime.Now.Date;

    try

    {

        logger.Information($"Processing SQL Server Instance: {serviceName} ({sqlServerModel.MachineName})");

        foreach (var repository in _repositories)

        {

            string friendlyName = string.Concat(repository.FriendlyName.Where(c => !char.IsWhiteSpace(c)));

            string loadName = $"RockwellMfgIndex{friendlyName}:{serviceName}";

            var lastDate = loadStateRepository.GetDateFor(loadName);

            lastDate = lastDate.AddDays(1);

```

```
if (lastDate < _minDate)
```

```
{
```

```
    lastDate = _minDate;
```

```
}
```

```
if (lastDate >= now)
```

```
{
```

```
    logger.Information($"Skipping extraction for '{serviceName}' as it's already processed through
```

```
'{lastDate:yyyy-MM-dd}'.");
```

```
    continue;
```

```
}
```

```
List<RockwellMfgIndexModel> transactions = await repository.GetAllAfterDate(serverName, lastDate,  
cancellationToken);
```

```
logger.Information($"Server: {serviceName}, Repo: {repository.FriendlyName}, Loading from: {lastDate},
```

```
Count: {transactions.Count}");
```

```
foreach (var transaction in transactions)
```

```
{
```

```
    nScaleTransaction.Results.Add(transaction);
```

```
}
```

```
nScaleTransaction.LoadStateUpdates.Add(loadName, lastDate);
```

```
if (!nScaleTransaction.Results.Any())
```

```
{
```

```

        nScaleTransaction.Exceptions.Add(new EmptyResultsException());

    }

}

catch (Exception e)

{

    if (e.IsNetworkConnectionException())

    {

        logger.Information($"'{serviceName}' is currently unavailable. Skipping extraction for this execution.");

    }

    logger.Information($"An error occurred while extracting data from {serviceName}");

    e.Data.Add("X-Server", server);

    nScaleTransaction.Exceptions.Add(e);

}

return nScaleTransaction;

}

}

}

```

NScaleTransaction.cs (Monitoring.ETL.Domain\RockwellMfgIndex\RMQLoader\NScaleTransaction.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
namespace Monitoring.ETL.Domain.RockwellMfgIndex.RMQLoader
```

```

{

    public class NScaleTransaction

    {

        public NScaleTransaction()

        {

            Results = new List<RockwellMfgIndexModel>();

            Exceptions = new List<Exception>();

            LoadStateUpdates = new Dictionary<string, DateTime>();

        }

        public List<RockwellMfgIndexModel> Results { get; }

        public List<Exception> Exceptions { get; }

        public Dictionary<string, DateTime> LoadStateUpdates { get; }

    }

}

```

Repository.cs (Monitoring.ETL.Domain\RockwellMfgIndex\RMQLoader\Repository.cs):

```

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Threading;

using System.Threading.Tasks;

namespace Monitoring.ETL.Domain.RockwellMfgIndex.RMQLoader

{

    internal class Repository

```

```

{

    private readonly string _database;

    private readonly string _commandText;

    private readonly int _commandTimeout;

    private const string SqlConnectionStringTemplate = "Data Source={0};Initial Catalog={1};Integrated
Security=True;";

    public Repository(string database, string commandText, string friendlyName, int commandTimeout = 600)
    {

        _database = database;

        _commandText = commandText;

        FriendlyName = friendlyName;

        _commandTimeout = commandTimeout;

    }

    public string FriendlyName { get; set; }

    public async Task<List<RockwellMfgIndexModel>> GetAllAfterDate(string serverName, DateTime lastDate,
Cancellation token cancellationToken)

    {

        var transactions = new List<RockwellMfgIndexModel>();

        using (var connection = new SqlConnection(string.Format(SqlConnectionStringTemplate, serverName,
_database)))

            using (var command = connection.CreateCommand())

            {

```

```
command.CommandType = System.Data.CommandType.Text;

command.CommandTimeout = _commandTimeout;


var lastDateParam = command.CreateParameter();

lastDateParam.ParameterName = "@Last_Date";

lastDateParam.DbType = System.Data.DbType.DateTime;

lastDateParam.Direction = System.Data.ParameterDirection.Input;

lastDateParam.Value = lastDate;


command.Parameters.Add(lastDateParam);


command.CommandText = _commandText;


await connection.OpenAsync(cancellationToken);

var reader = await command.ExecuteReaderAsync(cancellationToken);


if (reader.HasRows)

{

    while (await reader.ReadAsync(cancellationToken))

    {

        transactions.Add(new RockwellMfgIndexModel

        {

            JsonMessage = await Get<string>(reader, 0)

        });

    }

}
```

```

        connection.Close();

    }

    return transactions;
}

private async Task<T> Get<T>(SqlDataReader reader, int ordinal)
{
    try
    {
        return (await reader.IsDBNullAsync(ordinal)) ? default(T) : (T)reader.GetValue(ordinal);
    }
    catch (Exception)
    {
        return default(T);
    }
}
}
}

```

RepositoryLoader.cs (Monitoring.ETL.Domain\RockwellMfgIndex\RMQLoader\RepositoryLoader.cs):

```

using System.Collections.Generic;

using System.Configuration;

using System.Linq;

```



```
namespace Monitoring.ETL.Domain.RockwellMfgIndex.RMQLoader
```

```
{  
  
    internal static class RepositoryLoader  
  
    {  
  
        public static List<Repository> Load()  
  
        {  
  
            var repositories = new List<Repository>();  
  
  
            var repos = ConfigurationManager.AppSettings["RMI_Repositories"].Split(',');  
  
  
            if (repos.Contains("ProductionRate"))  
  
                repositories.Add(GetProductionRateRepository());  
  
  
            if (repos.Contains("ProductionRecord"))  
  
                repositories.Add(GetProductionRecordRepository());  
  
  
            if (repos.Contains("ReleaseStatus"))  
  
                repositories.Add(GetReleaseStatusRepository());  
  
  
            if (repos.Contains("MaterialMROCost"))  
  
                repositories.Add(GetMaterialMroCostRepository());  
  
  
            if (repos.Contains("InventoryDaysOnHand"))  
  
                repositories.Add(GetInventoryDaysOnHandRepository());  
  
  
            return repositories;  
  
        }  
    }  
}
```

```
}
```

```
private static Repository GetInventoryDaysOnHandRepository()
```

```
{
```

```
    const string database = "Part";
```

```
    const string commandText = @"
```

```
        USE Part;
```

```
        SET NOCOUNT ON;
```

```
        SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

```
        DECLARE
```

```
            @Shipper_Start_Date DATETIME = DATEADD(WEEK, -4, @Last_Date),
```

```
            @PCN INT,
```

```
            @RAW_USD_Amount DECIMAL(19, 5),
```

```
            @WIP_USD_Amount DECIMAL(19, 5),
```

```
            @FG_USD_Amount DECIMAL(19, 5),
```

```
            @COGS_USD_Amount DECIMAL(19, 5);
```

```
        DROP TABLE IF EXISTS dbo.#Container_History;
```

```
        DROP TABLE IF EXISTS dbo.#DOH_Results;
```

```
        CREATE TABLE dbo.#Container_History
```

```
        (
```

```
            Serial_No VARCHAR(25) NOT NULL,
```

```
            Change_Key BIGINT NOT NULL,
```

```
            Part_Key INT NULL,
```

```
Part_Operation_Key INT NULL,  
  
Material_Key INT NULL,  
  
Quantity DECIMAL(19, 5) NULL,  
  
Suboperation INT NULL,  
  
Finished_Goods INT NULL,  
  
Supply_Inventory_Type INT NULL  
);
```

```
CREATE TABLE dbo.#DOH_Results
```

```
(  
  
    PCN INT NOT NULL,  
  
    RAW_USD_Amount DECIMAL(19, 5) NULL,  
  
    WIP_USD_Amount DECIMAL(19, 5) NULL,  
  
    FG_USD_Amount DECIMAL(19, 5) NULL,  
  
    COGS_USD_Amount DECIMAL(19, 5) NULL,  
  
    PRIMARY KEY (PCN)  
);
```

```
INSERT dbo.#DOH_Results
```

```
(  
  
    PCN  
  
)
```

```
SELECT
```

```
    RPC.PCN
```

```
FROM Plexus_System.dbo.Resident_Plexus_Customer AS RPC
```

```
JOIN Common.dbo.Customer AS C
```

```

ON C.Plexus_Customer_No = 1

AND C.Customer_No = RPC.PCN

AND (C.Customer_Type = 'POL' OR C.Customer_Type = 'PMC Implementing');


DECLARE RPC_PCN_Cursor CURSOR FAST_FORWARD FOR

SELECT

    RPC.PCN

FROM dbo.#DOH_Results AS RPC;


OPEN RPC_PCN_Cursor;

FETCH NEXT FROM RPC_PCN_Cursor INTO @PCN;


WHILE @@FETCH_STATUS = 0

BEGIN

    DECLARE

        @Cost_Model_Key INT,

        @Currency_Conversion DECIMAL(19, 5) = 1.0;


    IF NOT EXISTS

        (

            SELECT

                C.Currency_Code

            FROM Plexus_Control.dbo.Plexus_Customer AS PC

            JOIN Common.dbo.Currency AS C

                ON C.Currency_Key = PC.Currency_Key

            AND C.Currency_Code = 'USD'

```

```

WHERE PC.Plexus_Customer_No = @PCN

)

BEGIN

SELECT

    @Currency_Conversion = COALESCE(1/ISNULL(UNCO.Conversion, UNCO2.Conversion),

CUR.Exchange_Rate, 1)

FROM Plexus_Control.dbo.Plexus_Customer AS PC

JOIN Common.dbo.Currency AS C

    ON C.Currency_Key = PC.Currency_Key

OUTER APPLY

(

    SELECT TOP(1)

        UC1.Conversion

    FROM Common.dbo.Unit_Conversion AS UC1

    WHERE UC1.Plexus_Customer_No = PC.Plexus_Customer_No

        AND UC1.Unit1 = C.Currency_Code

        AND UC1.Unit2 = 'USD'

    ) AS UNCO

OUTER APPLY

(

    SELECT TOP(1)

        1/ISNULL(NULLIF(UC2.Conversion, 0), 1) AS Conversion

    FROM Common.dbo.Unit_Conversion AS UC2

    WHERE UC2.Plexus_Customer_No = PC.Plexus_Customer_No

        AND UC2.Unit1 = 'USD'

        AND UC2.Unit2 = C.Currency_Code

```

) AS UNCO2

OUTER APPLY

(

SELECT

CC.Exchange_Rate/C.Exchange_Rate AS Exchange_Rate,

CC.Currency_HTML

FROM Common.dbo.Currency AS CC

WHERE CC.Currency_Code = C.Currency_Code

) AS CUR

WHERE PC.Plexus_Customer_No = @PCN

END;

SELECT TOP(1)

@Cost_Model_Key = CM.Cost_Model_Key

FROM dbo.Cost_Model AS CM

WHERE CM.PCN = @PCN

AND CM.Primary_Model = 1

AND CM.Deleted = 0;

WITH Max_Change_Keys AS

(

SELECT

CCM.Serial_No,

MAX(CCM.Change_Key) AS Change_Key

FROM dbo.Container_Change2 AS CCM WITH (INDEX=IX_History1)

WHERE CCM.Plexus_Customer_No = @PCN

AND CAST(CCM.Change_Date AS DATE) <= @Last_Date

GROUP BY CCM.Serial_No

),

Active_Max_Change_Keys AS

(

SELECT

MCK.Serial_No,

MCK.Change_Key

FROM Max_Change_Keys AS MCK

WHERE NOT EXISTS

(

SELECT

CC.Change_Key

FROM dbo.Container_Change2 AS CC

WHERE CC.Plexus_Customer_No = @PCN

AND CC.Active = 0

AND CC.Serial_No = MCK.Serial_No

AND CC.Change_Key = MCK.Change_Key

)

)

INSERT dbo.#Container_History WITH(TABLOCK)

(

Serial_No,

Change_Key,

Part_Key,

Part_Operation_Key,

Material_Key,
Quantity,
Suboperation,
Finished_Goods,
Supply_Inventory_Type

)

SELECT

AMCK.Serial_No,
AMCK.Change_Key,
CC.Part_Key,
CC.Part_Operation_Key,
CC.Material_Key,
CC.Quantity,
PO.Suboperation,
IT.Finished_Goods,
IT.Supply_Inventory_Type

FROM Active_Max_Change_Keys AS AMCK

JOIN dbo.Container_Change2 AS CC WITH

(FORCESEEK(PK_Container_Change2_PT(Plexus_Customer_No, Serial_No, Change_Key)))

ON CC.Plexus_Customer_No = @PCN

AND CC.Serial_No = AMCK.Serial_No

AND CC.Change_Key = AMCK.Change_Key

JOIN dbo.Container_Status AS CS

ON CS.Plexus_Customer_No = @PCN

AND CS.Container_Status = CC.Container_Status

AND CS.OK_Status = 1


```

LEFT OUTER JOIN dbo.Part_Operation AS PO WITH
(FORCESEEK(PK_Job_Operation(Plexus_Customer_No,Part_Key,Part_Operation_Key)))

    ON PO.Plexus_Customer_No = @PCN

    AND PO.Part_Operation_Key = CC.Part_Operation_Key

    AND PO.Part_Key = CC.Part_Key

LEFT OUTER JOIN dbo.Operation AS O WITH (FORCESEEK(PK_Operation(Plexus_Customer_No,
Operation_Key)))

    ON O.Plexus_Customer_No = PO.Plexus_Customer_No

    AND O.Operation_Key = PO.Operation_Key

LEFT OUTER JOIN dbo.Inventory_Type AS IT WITH
(FORCESEEK(PK_Inventory_Type(Plexus_Customer_No, Inventory_Type)))

    ON IT.Plexus_Customer_No = O.Plexus_Customer_No

    AND IT.Inventory_Type = O.Inventory_Type

OPTION

(

    HASH GROUP,

    FORCE ORDER,

    HASH JOIN,

    LOOP JOIN,

    MAXDOP 8,

    MIN_GRANT_PERCENT=100,

    RECOMPILE

);

WITH Material_Containers AS

(

```

SELECT

SUM(ISNULL(CH.Quantity * MC.Unit_Price, 0)) AS RAW_USD_Amount

FROM dbo.#Container_History AS CH

JOIN Material.dbo.Material AS M

ON M.Plexus_Customer_No = @PCN

AND M.Material_Key = CH.Material_Key

CROSS APPLY

(

SELECT TOP(1)

PRC.Unit_Price

FROM Purchasing.dbo.Line_Item AS LI

JOIN Purchasing.dbo.Receipt AS PRC

ON PRC.Plexus_Customer_No = LI.Plexus_Customer_No

AND PRC.Line_Item_Key = LI.Line_Item_Key

AND PRC.Receive_Date < @Last_Date

WHERE LI.Plexus_Customer_No = @PCN

AND LI.Material_Key = CH.Material_Key

ORDER BY PRC.Receive_Date DESC

) AS MC

WHERE NULLIF(CH.Part_Key, 0) IS NULL

AND CH.Finished_Goods != 1

AND CH.Supply_Inventory_Type != 1

),

WIP_Containers AS

(

SELECT

```

SUM(ISNULL(CH.Quantity * POC.Cost, 0)) AS WIP_USD_Amount

FROM dbo.#Container_History AS CH

JOIN dbo.Part_Operation_Cost AS POC

ON POC.PCN = @PCN

AND POC.Cost_Model_Key = @Cost_Model_Key

AND POC.Part_Operation_Key = CH.Part_Operation_Key

AND POC.Part_Key = CH.Part_Key

WHERE NULLIF(CH.Part_Key, 0) IS NOT NULL

AND CH.Suboperation = 0

AND CH.Finished_Goods != 1

AND CH.Supply_Inventory_Type != 1

),

FG_Containers AS

(

SELECT

SUM(ISNULL(CH.Quantity * POC.Cost, 0)) AS FG_USD_Amount

FROM dbo.#Container_History AS CH

JOIN dbo.Part_Operation_Cost AS POC

ON POC.PCN = @PCN

AND POC.Cost_Model_Key = @Cost_Model_Key

AND POC.Part_Operation_Key = CH.Part_Operation_Key

AND POC.Part_Key = CH.Part_Key

WHERE NULLIF(CH.Part_Key, 0) IS NOT NULL

AND CH.Suboperation = 0

AND CH.Finished_Goods = 1

),

```

Shipper_Containers AS

(

SELECT

SL.Part_Key,

SUM(SC.Quantity) AS Quantity

FROM Sales.dbo.Shipper AS S

JOIN Sales.dbo.Shipper_Status AS SS

ON SS.PCN = S.PCN

AND SS.Shipper_Status_Key = S.Shipper_Status_Key

AND SS.Shipped = 1

JOIN Sales.dbo.Shipper_Line AS SL

ON SL.PCN = S.PCN

AND SL.Shipper_Key = S.Shipper_Key

JOIN dbo.Part AS P

ON P.Plexus_Customer_No = SL.PCN

AND P.Part_Key = SL.Part_Key

JOIN Sales.dbo.Shipper_Container AS SC

ON SC.PCN = SL.PCN

AND SC.Shipper_Line_Key = SL.Shipper_Line_Key

JOIN dbo.Container AS C

ON C.Plexus_Customer_No = SC.PCN

AND C.Serial_No = SC.Serial_No

WHERE S.PCN = @PCN

AND S.Ship_Date >= @Shipper_Start_Date

AND CAST(S.Ship_Date AS DATE) <= @Last_Date

GROUP BY SL.Part_Key

),

Max_Part_Operations AS

(

SELECT

SC.Part_Key,

ISNULL(Max_Op.Part_Operation_Key, -1) AS Part_Operation_Key,

SC.Quantity

FROM Shipper_Containers AS SC

OUTER APPLY

(

SELECT TOP(1)

PO.Part_Operation_Key

FROM dbo.Part_Operation AS PO

JOIN dbo.Part_Op_Type AS POT

ON POT.PCN = PO.Plexus_Customer_No

AND POT.Part_Op_Type_Key = PO.Part_Op_Type_Key

WHERE PO.Plexus_Customer_No = @PCN

AND PO.Part_Key = SC.Part_Key

AND PO.Active = 1

AND PO.Suboperation = 0

AND POT.[Standard] = 1

ORDER BY

PO.Operation_No DESC

) AS Max_Op

),

PCN_Customer_Inventory_Usage AS

```

(
SELECT
    P.Part_Key,
    PO.Part_Operation_Key,
    SUM
    (
        ISNULL
        (
            S.Quantity /
            CASE
                WHEN S.Conversion < 0 AND PO.Net_Weight != 0 THEN ABS(S.Conversion *
PO.Net_Weight)
                WHEN S.Conversion = 0 OR (S.Conversion < 0 AND PO.Net_Weight = 0) THEN 1
                ELSE S.Conversion
            END,
            0
        )
    ) AS Quantity
FROM
    (
        SELECT
            CIU.PCN,
            CIU.Quantity,
            SL.Conversion,
            SL.Part_Key
        FROM dbo.Customer_Inventory_Usage AS CIU

```

CROSS APPLY

(

SELECT TOP(1)

SL2.Conversion,

SL2.Part_Key

FROM Sales.dbo.Shipper_Container AS SC

JOIN Sales.dbo.Shipper_Line AS SL2

ON SL2.PCN = SC.PCN

AND SL2.Shipper_Line_Key = SC.Shipper_Line_Key

WHERE SC.PCN = CIU.PCN

AND SC.Serial_No = CIU.Serial_No

ORDER BY

SL2.Shipper_Line_Key DESC

) AS SL

WHERE CIU.PCN = @PCN

AND CIU.Usage_Date >= @Shipper_Start_Date

AND CAST(CIU.Usage_Date AS DATE) <= @Last_Date

AND CIU.Unuse = 0

) AS S

JOIN dbo.Part AS P

ON P.Plexus_Customer_No = S.PCN

AND P.Part_Key = S.Part_Key

CROSS APPLY

(

SELECT TOP(1)

PO2.Part_Operation_Key,

```

        PO2.Net_Weight

FROM dbo.Part_Operation AS PO2

JOIN dbo.Part_Op_Type AS POT

    ON POT.PCN = PO2.Plexus_Customer_No

    AND POT.Part_Op_Type_Key = PO2.Part_Op_Type_Key

WHERE PO2.Plexus_Customer_No = P.Plexus_Customer_No

    AND PO2.Part_Key = P.Part_Key

    AND PO2.Active = 1

    AND PO2.Suboperation = 0

    AND POT.[Standard] = 1

ORDER BY

    PO2.Operation_No DESC

) AS PO

GROUP BY

    P.Part_Key,

    PO.Part_Operation_Key

),

Shipped_Containers AS

(

    SELECT

        MPO.Part_Key,

        MPO.Part_Operation_Key,

        MPO.Quantity

    FROM Max_Part_Operations AS MPO

UNION ALL

    SELECT

```



```

        PCIU.Part_Key,

        PCIU.Part_Operation_Key,

        PCIU.Quantity

FROM PCN_Customer_Inventory_Usage AS PCIU

),

Part_Costs AS

(

    SELECT

        CSTB_SUB.Part_Key,

        CSTB_SUB.Part_Operation_Key,

        SUM(CSTB_SUB.Cost) AS Cost

    FROM Shipped_Containers AS SC

    JOIN Common.dbo.Cost_Type AS CT_SUB

        ON CT_SUB.PCN = @PCN

        AND CT_SUB.COGS_Column = 1

    JOIN Common.dbo.Cost_Sub_Type AS CST_SUB

        ON CST_SUB.PCN = CT_SUB.PCN

        AND CST_SUB.Cost_Type_Key = CT_SUB.Cost_Type_Key

    JOIN dbo.Cost_Sub_Type_Breakdown AS CSTB_SUB

        ON CSTB_SUB.PCN = CST_SUB.PCN

        AND CSTB_SUB.Part_Key = SC.Part_Key

        AND CSTB_SUB.Part_Operation_Key = SC.Part_Operation_Key

        AND CSTB_SUB.Cost_Model_Key = @Cost_Model_Key

        AND CSTB_SUB.Cost_Sub_Type_Key = CST_SUB.Cost_Sub_Type_Key

    GROUP BY

        CSTB_SUB.Part_Key,

```

```

        CSTB_SUB.Part_Operation_Key

    ),

    COGS_Containers AS

    (

        SELECT

            SUM(ISNULL(SC.Quantity * PC.Cost, 0)) AS COGS_USD_Amount

        FROM Shipped_Containers AS SC

        JOIN Part_Costs AS PC

            ON PC.Part_Key = SC.Part_Key

            AND PC.Part_Operation_Key = SC.Part_Operation_Key

    )

    SELECT

        @RAW_USD_Amount =

        (

            SELECT TOP(1)

                ISNULL(MC.RAW_USD_Amount/@Currency_Conversion, 0)

            FROM Material_Containers AS MC

        ),

        @WIP_USD_Amount =

        (

            SELECT TOP(1)

                ISNULL(WC.WIP_USD_Amount/@Currency_Conversion, 0)

            FROM WIP_Containers AS WC

        ),

        @FG_USD_Amount =

        (

```

```

SELECT TOP(1)

        ISNULL(FC.FG_USD_Amount/@Currency_Conversion, 0)

FROM FG_Containers AS FC

),

@COGS_USD_Amount =

(

        SELECT TOP(1)

                ISNULL(CC.COGS_USD_Amount/@Currency_Conversion, 0)

        FROM COGS_Containers AS CC

)

OPTION

(

        FORCE ORDER,

        HASH JOIN,

        LOOP JOIN,

        MAXDOP 8,

        RECOMPILE

);

```

```

TRUNCATE TABLE dbo.#Container_History;

```

```

UPDATE DR

```

```

SET

```

```

        DR.RAW_USD_Amount = @RAW_USD_Amount,

```

```

        DR.WIP_USD_Amount = @WIP_USD_Amount,

```

```

        DR.FG_USD_Amount = @FG_USD_Amount,

```

```

        DR.COGS_USD_Amount = @COGS_USD_Amount

FROM dbo.#DOH_Results AS DR

WHERE DR.PCN = @PCN;


    FETCH NEXT FROM RPC_PCN_Cursor INTO @PCN;

END;


CLOSE RPC_PCN_Cursor;

DEALLOCATE RPC_PCN_Cursor;


SELECT
(
    SELECT
        DR.PCN,

        DR.RAW_USD_Amount,

        DR.WIP_USD_Amount,

        DR.FG_USD_Amount,

        DR.COGS_USD_Amount,

        @Last_Date AS Record_Date,

        @@SERVERNAME AS SQL_Server

    FOR JSON PATH,

    WITHOUT_ARRAY_WRAPPER

) AS JSON_Message

FROM dbo.#DOH_Results AS DR

WHERE NULLIF(DR.RAW_USD_Amount, 0) IS NOT NULL

    OR NULLIF(DR.WIP_USD_Amount, 0) IS NOT NULL

```

```
OR NULLIF(DR.FG_USD_Amount, 0) IS NOT NULL
```

```
OR NULLIF(DR.COGS_USD_Amount, 0) IS NOT NULL;
```

```
DROP TABLE dbo.#DOH_Results;
```

```
DROP TABLE dbo.#Container_History;;
```

```
return new Repository(database, commandText, "Inventory Days on Hand", 5400);
```

```
}
```

```
private static Repository GetMaterialMroCostRepository()
```

```
{
```

```
    const string database = "Purchasing";
```

```
    const string commandText = @"
```

```
        USE Purchasing;
```

```
        SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

```
        WITH RPC_Receipts AS
```

```
        (
```

```
            SELECT
```

```
                R.Plexus_Customer_No AS PCN,
```

```
                L.Material_Key,
```

```
                L.Item_Key,
```

```
                ISNULL(R.Quantity * R.Price_Conversion * R.Unit_Price, 0) AS Total_Cost,
```

```
                CASE WHEN PO.Currency_Code = 'USD'
```

```
                    THEN 1
```

```
                    ELSE COALESCE(1/ISNULL(UNCO.Conversion, UNCO2.Conversion), CUR.Exchange_Rate, 1)
```

```

END AS Currency_Conversion

FROM Plexus_System.dbo.Resident_Plexus_Customer AS RPC

JOIN dbo.Receipt AS R

    ON R.Plexus_Customer_No = RPC.PCN

    AND CAST(R.Receive_Date AS DATE) = @Last_Date

JOIN dbo.Line_Item AS L

    ON L.Plexus_Customer_No = R.Plexus_Customer_No

    AND L.Line_Item_Key= R.Line_Item_Key

    AND

    (

        NULLIF(L.Material_Key, 0) IS NOT NULL

        OR NULLIF(L.Item_Key, 0) IS NOT NULL

    )

JOIN dbo.PO AS PO

    ON PO.Plexus_Customer_No = L.Plexus_Customer_No

    AND PO.PO_Key = L.PO_Key

JOIN Plexus_Control.dbo.Plexus_Customer AS PC

    ON PC.Plexus_Customer_No = RPC.PCN

JOIN Common.dbo.Currency AS C

    ON C.Currency_Key = PC.Currency_Key

OUTER APPLY

(

    SELECT TOP(1)

        UC1.Conversion

    FROM Common.dbo.Unit_Conversion AS UC1

    WHERE UC1.Plexus_Customer_No = PO.Plexus_Customer_No

```

AND UC1.Unit1 = PO.Currency_Code

AND UC1.Unit2 = 'USD'

) AS UNCO

OUTER APPLY

(

SELECT TOP(1)

1/ISNULL(NULLIF(UC2.Conversion, 0), 1) AS Conversion

FROM Common.dbo.Unit_Conversion AS UC2

WHERE UC2.Plexus_Customer_No = PO.Plexus_Customer_No

AND UC2.Unit1 = 'USD'

AND UC2.Unit2 = PO.Currency_Code

) AS UNCO2

OUTER APPLY

(

SELECT

CC.Exchange_Rate/C.Exchange_Rate AS Exchange_Rate,

CC.Currency_HTML

FROM Common.dbo.Currency AS CC

WHERE CC.Currency_Code = PO.Currency_Code

) AS CUR

),

Grouped_Material_Receipts AS

(

SELECT

RR.PCN,

SUM(RR.Total_Cost/RR.Currency_Conversion) AS Direct_Material_Cost

```

FROM RPC_Receipts AS RR

CROSS APPLY

(

    SELECT TOP(1)

        PM.Part_Material_Key

    FROM Part.dbo.Part_Material AS PM

    WHERE PM.Plexus_Customer_No = RR.PCN

        AND PM.Material_Key = RR.Material_Key

    ) AS PM

WHERE NULLIF(RR.Material_Key, 0) IS NOT NULL

GROUP BY RR.PCN

),

Grouped_MRO_Receipts AS

(

    SELECT

        RR.PCN,

        SUM(RR.Total_Cost/RR.Currency_Conversion) AS MRO_Item_Cost

    FROM RPC_Receipts AS RR

    WHERE NULLIF(RR.Item_Key, 0) IS NOT NULL

    GROUP BY RR.PCN

),

Grouped_PCN AS

(

    SELECT

        GMR.PCN

    FROM Grouped_Material_Receipts AS GMR

```



```

UNION

SELECT

    GMR2.PCN

FROM Grouped_MRO_Receipts AS GMR2

),

Results AS

(

    SELECT

        GP.PCN,

        GMR.Direct_Material_Cost,

        GMR2.MRO_Item_Cost

    FROM Grouped_PCN AS GP

    LEFT OUTER JOIN Grouped_Material_Receipts AS GMR

        ON GMR.PCN = GP.PCN

    LEFT OUTER JOIN Grouped_MRO_Receipts AS GMR2

        ON GMR2.PCN = GP.PCN

    WHERE NULLIF(GMR.Direct_Material_Cost, 0) IS NOT NULL

        OR NULLIF(GMR2.MRO_Item_Cost, 0) IS NOT NULL

)

SELECT

(

    SELECT

        R.PCN,

        R.Direct_Material_Cost,

        R.MRO_Item_Cost,

        @Last_Date AS Record_Date,

```

```
@@SERVERNAME AS SQL_Server
```

```
FOR JSON PATH,
```

```
WITHOUT_ARRAY_WRAPPER
```

```
) AS JSON_Message
```

```
FROM Results AS R;"
```

```
return new Repository(database, commandText, "Material MRO Cost");
```

```
}
```

```
private static Repository GetProductionRateRepository()
```

```
{
```

```
const string database = "Part";
```

```
const string commandText = @"
```

```
USE Part;
```

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

```
WITH Results AS
```

```
(
```

```
SELECT
```

```
RPC.PCN,
```

```
SUM(COALESCE(JO.Quantity, J.Quantity)) AS Job_Scheduled_Qty,
```

```
SUM(P.Quantity) AS Job_Produced_Qty
```

```
FROM Plexus_System.dbo.Resident_Plexus_Customer AS RPC
```

```
JOIN dbo.Job AS J
```

```
ON J.PCN = RPC.PCN
```

```
JOIN dbo.Job_Status AS JS
```

```

ON JS.PCN = J.PCN

AND JS.Job_Status_Key = J.Job_Status_Key

AND (JS.Completed_Status = 1 OR JS.Active = 1)

JOIN dbo.Job_Op AS JO

ON JO.PCN = J.PCN

AND JO.Job_Key = J.Job_Key

AND CAST(JO.Complete_Date AS DATE) = @Last_Date

JOIN dbo.Job_Op_Status AS JOS

ON JOS.PCN = JO.PCN

AND JOS.Job_Op_Status_Key = JO.Job_Op_Status_Key

AND JOS.Completed_Status = 1

AND JOS.Not_Required_Status = 0

CROSS APPLY

(

SELECT

SUM(P.Quantity) AS Quantity

FROM dbo.Production AS P

WHERE P.Plexus_Customer_No = JO.PCN

AND P.Part_Key = JO.Part_Key

AND P.Job_Op_Key = JO.Job_Op_Key

GROUP BY

P.Plexus_Customer_No

) AS P

GROUP BY RPC.PCN

)

SELECT

```

```
(
SELECT
    R.PCN,
    R.Job_Scheduled_Qty,
    R.Job_Produced_Qty,
    @Last_Date AS Record_Date,
    @@SERVERNAME AS SQL_Server
FOR JSON PATH,
WITHOUT_ARRAY_WRAPPER
) AS JSON_Message
FROM Results AS R";
```

```
return new Repository(database, commandText, "Production Rate");
```

```
}
```

```
private static Repository GetProductionRecordRepository()
```

```
{
```

```
const string database = "Part";
```

```
const string commandText = @"
```

```
USE Part;
```

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

```
WITH Results AS
```

```
(
```

```
SELECT
```

```
P.Plexus_Customer_No AS PCN,
```

COUNT(*) AS Num_Production_Records

FROM (

SELECT

P1.Plexus_Customer_No

FROM Plexus_System.dbo.Resident_Plexus_Customer AS PC1

JOIN Part.dbo.Workcenter AS W1

ON W1.Plexus_Customer_No = PC1.PCN

JOIN Part.dbo.Production AS P1 WITH (FORCESEEK

(IX_Workcenter_Key_Record_Date(Plexus_Customer_No, Workcenter_Key)))

ON P1.Plexus_Customer_No = W1.Plexus_Customer_No

AND P1.Workcenter_Key = W1.Workcenter_Key

WHERE CAST(P1.Record_Date AS DATE) = @Last_Date

UNION ALL

SELECT

P2.Plexus_Customer_No

FROM Plexus_System.dbo.Resident_Plexus_Customer AS PC2

JOIN Part.dbo.Production AS P2 WITH (FORCESEEK

(IX_Workcenter_Key_Record_Date(Plexus_Customer_No, Workcenter_Key)))

ON P2.Plexus_Customer_No = PC2.PCN

WHERE P2.Workcenter_Key IS NULL

AND CAST(P2.Record_Date AS DATE) = @Last_Date

) AS P

GROUP BY P.Plexus_Customer_No

)

```
SELECT
```

```
(
```

```
    SELECT
```

```
        R.PCN,
```

```
        R.Num_Production_Records,
```

```
        @Last_Date AS Record_Date,
```

```
        @@SERVERNAME AS SQL_Server
```

```
    FOR JSON PATH,
```

```
    WITHOUT_ARRAY_WRAPPER
```

```
) AS JSON_Message
```

```
FROM Results AS R";
```

```
return new Repository(database, commandText, "Production Records");
```

```
}
```

```
private static Repository GetReleaseStatusRepository()
```

```
{
```

```
    const string database = "Sales";
```

```
    const string commandText = @"
```

```
        USE Sales;
```

```
        SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

```
        DROP TABLE IF EXISTS dbo.#RPC_Releases;
```

```
        CREATE TABLE dbo.#RPC_Releases
```

```
(
```

```
PCN INT NOT NULL,  
  
Release_Key INT NOT NULL,  
  
Due_Date DATETIME NULL,  
  
Ship_Date DATETIME NULL,  
  
Quantity DECIMAL(18) NULL  
  
);
```

```
INSERT dbo.#RPC_Releases WITH (TABLOCK)
```

```
(  
  
    PCN,  
  
    Release_Key,  
  
    Due_Date,  
  
    Ship_Date,  
  
    Quantity  
  
)
```

```
SELECT
```

```
    R.PCN,  
  
    R.Release_Key,  
  
    R.Due_Date,  
  
    R.Ship_Date,  
  
    R.Quantity
```

```
FROM Plexus_System.dbo.Resident_Plexus_Customer AS RPC
```

```
JOIN dbo.Release AS R
```

```
    ON R.PCN = RPC.PCN
```

```
    AND R.Add_Date < @Last_Date
```

```
AND
```

```

(
    R.Ship_Date IS NULL

    OR R.Ship_Date >= @Last_Date
)

JOIN dbo.Release_Type AS RT

    ON RT.PCN = RPC.PCN

    AND RT.Release_Type_Key = R.Release_Type_Key

    AND RT.Firm = 1

JOIN dbo.Release_Status AS RS

    ON RS.PCN = RPC.PCN

    AND RS.Release_Status_Key = R.Release_Status_Key

    AND

(
    RS.Active = 1

    OR RS.Shipped_Status = 1
)

OPTION(RECOMPILE, FORCE ORDER, LOOP JOIN, MAXDOP 8);

DELETE FROM dbo.#RPC_Releases

WHERE Due_Date IS NULL;

WITH RPC_Num_Scheduled AS

(
    SELECT

        RR.PCN,

        COUNT(*) AS Num_Releases_Scheduled

```



```

FROM dbo.#RPC_Releases AS RR

GROUP BY RR.PCN

),

RPC_Num_Past_Due AS

(

    SELECT

        RR.PCN,

        COUNT(*) AS Num_Releases_Past_Due

    FROM dbo.#RPC_Releases AS RR

    WHERE RR.Due_Date < @Last_Date

    AND

    (

        RR.Ship_Date > RR.Due_Date

        OR ISNULL(

            (

                SELECT

                    SUM(C.Quantity) AS Qty_Shipped

                FROM dbo.Shipper_Container AS C

                JOIN dbo.Shipper_Line AS L

                    ON L.PCN = C.PCN

                    AND L.Shipper_Line_Key = C.Shipper_Line_Key

                JOIN dbo.Shipper AS S

                    ON S.PCN = L.PCN

                    AND S.Shipper_Key = L.Shipper_Key

                JOIN dbo.Shipper_Status AS SS

                    ON SS.PCN = S.PCN
            )
        )
    )

```

AND SS.Shipper_Status_Key = S.Shipper_Status_Key

AND SS.Shipped = 1

WHERE C.PCN = RR.PCN

AND C.Release_Key = RR.Release_Key

GROUP BY C.Release_Key

), 0) < RR.Quantity

)

GROUP BY RR.PCN

),

Results AS

(

SELECT

RNS.PCN,

RNS.Num_Releases_Scheduled,

RNPD.Num_Releases_Past_Due

FROM RPC_Num_Scheduled AS RNS

LEFT OUTER JOIN RPC_Num_Past_Due AS RNPD

ON RNPD.PCN = RNS.PCN

)

SELECT

(

SELECT

R.PCN,

R.Num_Releases_Scheduled,

R.Num_Releases_Past_Due,

@Last_Date AS Record_Date,

```

        @@SERVERNAME AS SQL_Server

        FOR JSON PATH,

        WITHOUT_ARRAY_WRAPPER

    ) AS JSON_Message

    FROM Results AS R

    OPTION(FORCE ORDER);

```

```

DROP TABLE dbo.#RPC_Releases;";

```

```

        return new Repository(database, commandText, "Release Status");
    }
}
}

```

RockwellMfgIndexModel.cs (Monitoring.ETL.Domain\RockwellMfgIndex\RMQLoader\RockwellMfgIndexModel.cs):

```

using ETL.Process;

```

```

namespace Monitoring.ETL.Domain.RockwellMfgIndex.RMQLoader
{
    public class RockwellMfgIndexModel : IExtractModel, SqlJson.IJsonExtractModel
    {
        public string JsonMessage { get; set; }
    }
}

```

Transformer.cs (Monitoring.ETL.Domain\RockwellMfgIndex\RMQLoader\Transformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.RockwellMfgIndex.RMQLoader
```

```
{
```

```
    public class Transformer : ITransformer<RockwellMfgIndexModel, RabbitMQ.Model.RockwellMfgIndex>
```

```
    {
```

```
        public async Task<IEnumerable<RabbitMQ.Model.RockwellMfgIndex>>
```

```
TransformAsync(IEnumerable<RockwellMfgIndexModel> extracted, CancellationToken cancellationToken,
```

```
IEtlProcessLogger logger)
```

```
    {
```

```
        await Task.Yield();
```

```
        return extracted.Select(pc =>
```

```
            new RabbitMQ.Model.RockwellMfgIndex
```

```
            {
```

```
                JsonMessage = pc.JsonMessage
```

```
            });
```

```
        }
```

```
    }
```

```
}
```

ServiceNowApiResults.cs (Monitoring.ETL.Domain\ServiceNowCmdB\Extract\ServiceNowApiResults.cs):

```
using System.Collections.Generic;
```

```
using Newtonsoft.Json;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdB.Extract
```

```
{
```

```
    /// <summary>
```

```
    /// Generic class to help deserialization of API results.
```

```
    /// </summary>
```

```
    /// <typeparam name="T"></typeparam>
```

```
    public sealed class ServiceNowApiResults<T>
```

```
    {
```

```
        [JsonProperty("result")]
```

```
        public List<T> Results { get; set; }
```

```
    }
```

```
}
```

ServiceNowConfigurationItemModel.cs

(Monitoring.ETL.Domain\ServiceNowCmdB\Extract\ServiceNowConfigurationItemModel.cs):

```
using Newtonsoft.Json;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdB.Extract
```

```
{
```

```
    /// <summary>
```

```
    /// POCO for generic Configuration Item object.
```

```
    /// </summary>
```

```
public class ServiceNowConfigurationItemModel
```

```
{
```

```
    [JsonProperty("sys_id")]
```

```
    public string SysId { get; set; }
```

```
    [JsonProperty("name")]
```

```
    public string Name { get; set; }
```

```
    [JsonProperty("sys_class_name")]
```

```
    public string SysClassName { get; set; }
```

```
}
```

```
}
```

ServiceNowFileSystemModel.cs (Monitoring.ETL.Domain\ServiceNowCmdb\Extract\ServiceNowFileSystemModel.cs):

```
using Newtonsoft.Json;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Extract
```

```
{
```

```
    public sealed class ServiceNowFilesystemModel
```

```
    {
```

```
        [JsonProperty("free_space")]
```

```
        public string FreeSpace { get; set; }
```

```
        [JsonProperty("size")]
```

```
        public string Size { get; set; }
```

```
[JsonProperty("mount_point")]
```

```
public string MountPoint { get; set; }
```

```
[JsonProperty("file_system")]
```

```
public string FileSystem { get; set; }
```

```
[JsonProperty("sys_id")]
```

```
public string SysId { get; set; }
```

```
[JsonProperty("name")]
```

```
public string Name { get; set; }
```

```
[JsonProperty("free_space_bytes")]
```

```
public long? FreeSpaceBytes { get; set; }
```

```
[JsonProperty("size_bytes")]
```

```
public long? SizeBytes { get; set; }
```

```
[JsonProperty("label")]
```

```
public string Label { get; set; }
```

```
[JsonProperty("computer")]
```

```
public string ComputerSysId { get; set; }
```

```
[JsonProperty("lun")]
```

```
public string Lun { get; set; }
```

```
[JsonProperty("device")]
```

```
public string Device { get; set; }
```

```
[JsonProperty("media_type")]
```

```
public string MediaType { get; set; }
```

```
[JsonProperty("provided_by")]
```

```
public string ProvidedBySysId { get; set; }
```

```
public ServiceNowConfigurationItemModel ProvidedBy { get; set; }
```

```
}
```

```
}
```

ServiceNowLocationModel.cs (Monitoring.ETL.Domain\ServiceNowCmdb\Extract\ServiceNowLocationModel.cs):

```
using Newtonsoft.Json;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Extract
```

```
{
```

```
public sealed class ServiceNowLocationModel
```

```
{
```

```
[JsonProperty("name")]
```

```
public string Name { get; set; }
```

```
[JsonProperty("sys_id")]
```

```
public string SysId { get; set; }
```



```
}
```

```
}
```

ServiceNowRelationshipModel.cs

(Monitoring.ETL.Domain\ServiceNowCmdb\Extract\ServiceNowRelationshipModel.cs):

```
using Newtonsoft.Json;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Extract
```

```
{
```

```
    public sealed class ServiceNowRelationshipModel
```

```
    {
```

```
        [JsonProperty("parent")]
```

```
        public string ParentSysId { get; set; }
```

```
        public ServiceNowConfigurationItemModel ParentConfigurationItem { get; set; }
```

```
        [JsonProperty("child")]
```

```
        public string ChildSysId { get; set; }
```

```
        [JsonProperty("type")]
```

```
        public string RelationshipTypeSysId { get; set; }
```

```
        public ServiceNowConfigurationItemModel ChildConfigurationItem { get; set; }
```

```
        [JsonProperty("port")]
```

```
        public string Port { get; set; }
```

```
[JsonProperty("sys_id")]
```

```
public string SysId { get; set; }
```

```
public ServiceNowRelationshipTypeModel RelationshipType { get; set; }
```

```
}
```

```
}
```

ServiceNowRelationshipTypeModel.cs

(Monitoring.ETL.Domain\ServiceNowCmdb\Extract\ServiceNowRelationshipTypeModel.cs):

```
using Newtonsoft.Json;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Extract
```

```
{
```

```
public sealed class ServiceNowRelationshipTypeModel
```

```
{
```

```
[JsonProperty("sys_id")]
```

```
public string SysId { get; set; }
```

```
[JsonProperty("name")]
```

```
public string Name { get; set; }
```

```
[JsonProperty("child_descriptor")]
```

```
public string ChildDescriptor { get; set; }
```

```
[JsonProperty("parent_descriptor")]
```

```
public string ParentDescriptor { get; set; }
```

```
[JsonProperty("sys_name")]
```

```
public string SysName { get; set; }
```

```
}
```

```
}
```

ServiceNowRestApiClient.cs (Monitoring.ETL.Domain\ServiceNowCmdb\Extract\ServiceNowRestApiClient.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.IO;
```

```
using System.Linq;
```

```
using System.Net;
```

```
using System.Text;
```

```
using System.Threading;
```

```
using Libraries.Common.Exceptions;
```

```
using Monitoring.ETL.Process;
```

```
using Newtonsoft.Json;
```

```
using Plex.Infrastructure.ConfigSystems;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Extract
```

```
{
```

```
public sealed class ServiceNowRestApiClient
```

```
{
```

```
private readonly IEtlProcessLogger _logger;
```

```
private const int DefaultBatchSize = 500;
```

```
private const string TableApiBaseUrl = "/api/now/table";

private readonly string _url;

private readonly string _userid;

private readonly string _password;


private List<ServiceNowRelationshipTypeModel> _relationshipTypes;


public ServiceNowRestApiClient(IEtlProcessLogger logger)

{

    _logger = logger;

    var registry = new PlexRegistrySystem();

    _userid = registry.GetString("CloudOps", "Monitoring", "ServiceNow", "ApiUserName", string.Empty);

    _password = registry.GetString("CloudOps", "Monitoring", "ServiceNow", "ApiPassword", string.Empty);

    _url = registry.GetString("CloudOps", "Monitoring", "ServiceNow", "ApiUrl", string.Empty);


    if (string.IsNullOrEmpty(_userid))

    {

        throw new StringNullOrEmptyException("The UserId value cannot be null or empty. The value is pulled from the

registry at HKLM\\Software\\Plex\\CloudOps\\Monitoring\\ServiceNow\\ApiUserName");

    }


    if (string.IsNullOrEmpty(_password))

    {

        throw new StringNullOrEmptyException("The Password value cannot be null or empty. The value is pulled from the

registry at HKLM\\Software\\Plex\\CloudOps\\Monitoring\\ServiceNow\\ApiPassword");

    }

}
```

```

        if (string.IsNullOrEmpty(_url))
        {
            throw new StringNullOrEmptyException("The Api url value cannot be null or empty. The value is pulled from the
registry at HKLM\\Software\\Plex\\CloudOps\\Monitoring\\ServiceNow\\ApiUrl");
        }
    }
}

```

```

private List<ServiceNowRelationshipTypeModel> GetRelationshipTypes(Cancellation_token cancellation_token)
{
    if (_relationshipTypes == null)
    {
        var sb = new StringBuilder()

        .Append(TableApiBaseUrl)

        .Append("/cmdb_rel_type")

        .Append("?sysparm_exclude_reference_link=true")

        .Append("&sysparm_suppress_pagination_header=true")

        .Append("&sysparm_limit=").Append(DefaultBatchSize)

        .Append("&sysparm_offset={0}")

        .Append("&sysparm_fields=child_descriptor,name,sys_id,sys_name,parent_descriptor");

        _relationshipTypes = GetResultsCollection<ServiceNowRelationshipTypeModel>(sb.ToString(),
cancellation_token);
    }

    if (cancellation_token.IsCancellationRequested)

```

```
{  
  
    return null;  
  
}
```

```
return _relationshipTypes;  
  
}
```

```
private List<ServiceNowRelationshipModel> GetRelationships(ICollection<string> computerIds,  
Cancellation token cancellationToken, string relationshipTypeFilter = null, bool includeParents = true)
```

```
{  
  
    if (computerIds.Count == 0)  
  
    {  
  
        return new List<ServiceNowRelationshipModel>();  
  
    }
```

```
var relationshipTypes = GetRelationshipTypes(cancellationToken);
```

```
if (cancellationToken.IsCancellationRequested)  
  
{  
  
    return null;  
  
}
```

```
var sb = new StringBuilder()  
  
    .Append(TableApiBaseUrl)  
  
    .Append("/cmdb_rel_ci")  
  
    .Append("?sysparm_exclude_reference_link=true")
```

```

.Append("&sysparm_suppress_pagination_header=true")

.Append("&sysparm_view=true")

.Append("&sysparm_limit=").Append(DefaultBatchSize)

.Append("&sysparm_offset={0}")

.Append("&sysparm_fields=parent,child,type,port,sys_id");

sb.Append("&sysparm_query=parent!N{1}");

if (includeParents)

{

    sb.Append("^ORchild!N{1}");

}

if (string.IsNullOrEmpty(relationshipTypeFilter) == false)

{

    var relationshipType = relationshipTypes.FirstOrDefault(rt => string.Equals(rt.SysName, relationshipTypeFilter,

StringComparison.OrdinalIgnoreCase));

    if (relationshipType != null)

    {

        sb.Append("^type=").Append(relationshipType.SysId);

    }

}

else

{

    // Be default, exclude the IP connection relationships. This adds a lot of noise for little value in out case.

    var ipConnectionRelationshipType = relationshipTypes.FirstOrDefault(rt => rt.SysName == "IP Connection::IP

```

```
Connection");
```

```
    if (ipConnectionRelationshipType != null)
```

```
    {
```

```
        sb.Append("^type!=")
```

```
        .Append(ipConnectionRelationshipType.SysId);
```

```
    }
```

```
}
```

```
var allRelationships = GetResultsCollection<ServiceNowRelationshipModel>(sb.ToString(),
```

```
SubdivideAndConcatList(computerIds, 60), cancellationToken);
```

```
if (cancellationToken.IsCancellationRequested)
```

```
{
```

```
    return null;
```

```
}
```

```
var sysIds = allRelationships
```

```
    .Select(r => r.ParentSysId)
```

```
    .Union(allRelationships.Select(r => r.ChildSysId))
```

```
    .Distinct()
```

```
    .ToList();
```

```
var configurationItems = GetConfigurationItems(sysIds, cancellationToken);
```

```
if (cancellationToken.IsCancellationRequested)
```



```

{

    return null;

}


foreach (var relationship in allRelationships)

{

    relationship.RelationshipType = relationshipTypes.FirstOrDefault(rt => string.Equals(rt.SysId,
relationship.RelationshipTypeSysId, StringComparison.OrdinalIgnoreCase));

    relationship.ParentConfigurationItem = configurationItems.FirstOrDefault(ci => string.Equals(ci.SysId,
relationship.ParentSysId, StringComparison.OrdinalIgnoreCase));

    relationship.ChildConfigurationItem = configurationItems.FirstOrDefault(ci => string.Equals(ci.SysId,
relationship.ChildSysId, StringComparison.OrdinalIgnoreCase));

}

return allRelationships.Where(r => r.ParentConfigurationItem != null && r.ChildConfigurationItem != null).ToList();

}


private List<ServiceNowConfigurationItemModel> GetConfigurationItems(ICollection<string> sysIds,
Cancellation token cancellationToken)

{

    if (sysIds.Count == 0)

    {

        return new List<ServiceNowConfigurationItemModel>();

    }

    var sb = new StringBuilder()

```

```

.Append(TableApiBaseUrl)

.Append("/cmdb_ci")

.Append("?sysparm_exclude_reference_link=true")

.Append("&sysparm_suppress_pagination_header=true")

.Append("&sysparm_limit=").Append(DefaultBatchSize)

.Append("&sysparm_offset={0}")

.Append("&sysparm_fields=name,sys_id,sys_class_name")

.Append("&sysparm_query=sys_idIN{1}");

```

```

return GetResultsCollection<ServiceNowConfigurationItemModel>(sb.ToString(), SubdivideAndConcatList(sysIds,
100), cancellationToken);

}

```

```

private List<ServiceNowWindowsFailoverAppModel> GetWindowsFailoverApplications(ICollection<string>
failoverClusterIds, CancellationToken cancellationToken)

{

    if (failoverClusterIds.Count == 0)

    {

        return new List<ServiceNowWindowsFailoverAppModel>();

    }

}

```

// This is a user defined table in service now.

// The "cluster node running" column is currently broken.

// Once it's fixed, we can show it on the interface.

```
var sb = new StringBuilder()
```

```
.Append(TableApiBaseUrl)
```

```

.Append("/u_cmdb_ci_win_cluster_failover_app")

.Append("?sysparm_exclude_reference_link=true")

.Append("&sysparm_suppress_pagination_header=true")

.Append("&sysparm_limit=").Append(DefaultBatchSize)

.Append("&sysparm_offset={0}")

.Append("&sysparm_fields=name,sys_id,sys_class_name,u_cluster,u_cluster_node_running")

.Append("&sysparm_query=u_active=true^u_cluster!N")

.Append(ToDelimitedListString(failoverClusterIds));

```

```

return GetResultsCollection<ServiceNowWindowsFailoverAppModel>(sb.ToString(),

```

```

SubdivideAndConcatList(failoverClusterIds, 100), cancellationToken);

```

```

}

```

```

private List<ServiceNowSoftwareInstallModel> GetSoftwareInstalls(IReadOnlyCollection<string> computerIds,

```

```

Cancellation token cancellationToken)

```

```

{

```

```

    if (computerIds == null)

```

```

    {

```

```

        throw new ArgumentNullException(nameof(computerIds));

```

```

    }

```

```

    if (computerIds.Count == 0)

```

```

    {

```

```

        return new List<ServiceNowSoftwareInstallModel>();

```

```

    }

```

```

var sb = new StringBuilder()

.Append(TableApiBaseUrl)

.Append("/cmdb_sam_sw_install")

.Append("?sysparm_exclude_reference_link=true")

.Append("&sysparm_suppress_pagination_header=true")

.Append("&sysparm_limit=").Append(DefaultBatchSize)

.Append("&sysparm_offset={0}")

.Append("&sysparm_fields=name,sys_id,installed_on,normalized_display_name,normalized_version,normalized_publisher")

.Append("&sysparm_query=installed_onIN{1}");

return GetResultsCollection<ServiceNowSoftwareInstallModel>(sb.ToString(),
SubdivideAndConcatList(computerIds, 100), cancellationToken);
}

private List<ServiceNowFilesystemModel> GetFileSystems(IReadOnlyCollection<string> computerIds,
Cancellation token cancellationToken)
{
    if (computerIds == null)
    {
        throw new ArgumentNullException(nameof(computerIds));
    }

    if (computerIds.Count == 0)
    {

```

```

        return new List<ServiceNowFilesystemModel>();
    }

    var sb = new StringBuilder()

        .Append(TableApiBaseUrl)

        .Append("/cmdb_ci_file_system")

        .Append("?sysparm_exclude_reference_link=true")

        .Append("&sysparm_suppress_pagination_header=true")

        .Append("&sysparm_limit=").Append(DefaultBatchSize)

        .Append("&sysparm_offset={0}")

        .Append("&sysparm_fields=name,sys_id,free_space,size,mount_point,file_system,free_space_bytes,size_bytes,label,computer,lun,device,provided_by,media_type")

        .Append("&sysparm_query=computerIN{1}");

    var results = GetResultsCollection<ServiceNowFilesystemModel>(sb.ToString(),
SubdivideAndConcatList(computerIds, 100), cancellationToken);

    if (cancellationToken.IsCancellationRequested)
    {
        return null;
    }

    var providedBySysIds = results

        .Select(fs => fs.ProvidedBySysId)

        .Where(s => string.IsNullOrEmpty(s) == false)

```

```
.Distinct()
```

```
.ToList();
```

```
if (providedBySysIds.Count > 0)
```

```
{
```

```
    var configItems = GetConfigurationItems(providedBySysIds, cancellationToken);
```

```
    if (cancellationToken.IsCancellationRequested)
```

```
    {
```

```
        return null;
```

```
    }
```

```
    results.ForEach(fs => fs.ProvidedBy = configItems.FirstOrDefault(ci => ci.SysId == fs.ProvidedBySysId));
```

```
}
```

```
return results;
```

```
}
```

```
private List<ServiceNowServerModel> GetAzureServers(Cancellation token cancellationToken, out
```

```
List<ServiceNowRelationshipModel> azureRelationships)
```

```
{
```

```
    _logger.Information("Getting Azure VMs");
```

```
    var sb = new StringBuilder()
```

```
        .Append(TableApiBaseUrl)
```

```
        .Append("/cmdb_ci_azure_vm")
```

```

.Append("?sysparm_exclude_reference_link=true")

.Append("&sysparm_suppress_pagination_header=true")

.Append("&sysparm_limit=").Append(DefaultBatchSize)

.Append("&sysparm_offset={0}")

.Append("&sysparm_fields=name,os,sys_id,sys_class_name,location")

.Append("&sysparm_query=u_active=true^sys_class_nameNOT
INcmdb_ci_hyper_v_server,cmdb_ci_isam_server");

```

```

var azureVMs = GetResultsCollection<ServiceNowServerModel>(sb.ToString(), cancellationToken);

```

```

var azureRelationshipsLocal = new List<ServiceNowRelationshipModel>();

```

```

// Create a fake host that will be used for all Azure VMs.

```

```

// The existence of a host is what the data model uses to determine if

```

```

// the server is a virtual machine.

```

```

var azureVmHost = new ServiceNowServerModel

```

```

{
    Name = "[Azure Hosting]",
    SysClassName = "none",
    SysId = Guid.Empty.ToString("N")
};

```

```

// Add relationship for all Azure VMs to the fake host created above.

```

```

var virtualizedByRelationshipType = GetRelationshipTypes(cancellationToken).FirstOrDefault(t =>
string.Equals(t.Name, "Virtualized by::Virtualizes", StringComparison.OrdinalIgnoreCase));

```

```
// Build the relationships for every azure VM to the "[Azure Hosting]" server.
```

```
azureVMs.ForEach(vm =>
```

```
{
```

```
    vm.Virtual = true;
```

```
    azureRelationshipsLocal.Add(new ServiceNowRelationshipModel
```

```
    {
```

```
        ChildConfigurationItem = azureVmHost,
```

```
        ChildSysId = azureVmHost.SysId,
```

```
        ParentConfigurationItem = vm,
```

```
        ParentSysId = vm.SysId,
```

```
        RelationshipTypeSysId = virtualizedByRelationshipType?.SysId,
```

```
        RelationshipType = virtualizedByRelationshipType
```

```
    });
```

```
});
```

```
azureVMs.Add(azureVmHost);
```

```
_logger.Information($"Azure VMs found: {azureVMs.Count}");
```

```
azureRelationships = azureRelationshipsLocal;
```

```
return azureVMs;
```

```
}
```

```
public IEnumerable<ServiceNowServerModel> GetServers(Cancellation_token cancellationToken)
```

```
{
```



```
_logger.Information($"Begin GetServers() Service Now Url: {_url}");
```

```
var sb = new StringBuilder()
```

```
.Append(TableApiBaseUrl)
```

```
.Append("/cmdb_ci_server")
```

```
.Append("?sysparm_exclude_reference_link=true")
```

```
.Append("&sysparm_suppress_pagination_header=true")
```

```
.Append("&sysparm_limit=").Append(DefaultBatchSize)
```

```
.Append("&sysparm_offset={0}")
```

```
.Append("&sysparm_fields=name,os,os_version,os_service_pack,ram,cpu_speed,cpu_count,cpu_core_count,cpu_core  
_thread,fqdn,sys_id,sys_class_name,virtual,location")
```

```
.Append("&sysparm_query=u_active=true^sys_class_nameNOT
```

```
INcmdb_ci_hyper_v_server,cmdb_ci_isam_server");
```

```
_logger.Information("Getting Servers");
```

```
var servers = GetResultsCollection<ServiceNowServerModel>(sb.ToString(), cancellationToken);
```

```
_logger.Information($"Servers Found: {servers.Count}");
```

```
if (cancellationToken.IsCancellationRequested)
```

```
{
```

```
    return null;
```

```
}
```

```
var azureVMs = GetAzureServers(cancellationToken, out var azureRelationships);
```

```
servers.AddRange(azureVMs);
```

```
if (cancellationToken.IsCancellationRequested)
```

```
{
```

```
    return null;
```

```
}
```

```
if (servers.Count > 0)
```

```
{
```

```
    var serverSysIds = servers.Select(s => s.SysId).ToList();
```

```
    _logger.Information("Getting Filesystems");
```

```
    var fileSystems = GetFileSystems(serverSysIds, cancellationToken);
```

```
    _logger.Information($"Filesystems Found: {fileSystems.Count}");
```

```
if (cancellationToken.IsCancellationRequested)
```

```
{
```

```
    return null;
```

```
}
```

```
_logger.Information("Getting Relationships");
```

```
var relationships = GetRelationships(serverSysIds, cancellationToken);
```

```
relationships.AddRange(azureRelationships);
```

```
_logger.Information($"Relationships Found: {relationships.Count}");
```

```
if (cancellationToken.IsCancellationRequested)
```

```
{  
  
    return null;  
  
}
```

```
_logger.Information("Getting Software Installs");  
  
var softwareInstances = GetSoftwareInstalls(serverSysIds, cancellationToken);  
  
_logger.Information($"Software Installs Found: {softwareInstances.Count}");
```

```
if (cancellationToken.IsCancellationRequested)  
  
{  
  
    return null;  
  
}
```

```
_logger.Information("Getting Locations");  
  
var locations = GetLocations(cancellationToken);  
  
_logger.Information($"Locations Found: {locations.Count}");
```

```
if (cancellationToken.IsCancellationRequested)  
  
{  
  
    return null;  
  
}
```

```
// The server is the child in the "Hosted on::Hosts" relationship.
```

```
bool IsFailoverCluster(ServiceNowRelationshipModel r) => string.Equals(r.RelationshipType.Name, "Hosted  
on::Hosts", StringComparison.OrdinalIgnoreCase)
```

```
    && string.Equals(r.ParentConfigurationItem.SysClassName,
```

```
"cmdb_ci_win_cluster_node", StringComparison.OrdinalIgnoreCase);
```

```
// Get the cluster noded ids so we can look up all failover clusters in a single batch.
```

```
var clusterNodeIds = relationships.Where(IsFailoverCluster)
```

```
.Select(r => r.ParentConfigurationItem.SysId)
```

```
.Distinct()
```

```
.ToList();
```

```
_logger.Information("Getting Failover Cluster Relationships");
```

```
var failoverClusterRelationships = GetFailoverClusterRelationships(clusterNodeIds, cancellationToken);
```

```
_logger.Information($"Failover Cluster Relationships Found: {failoverClusterRelationships.Count}");
```

```
if (cancellationToken.IsCancellationRequested)
```

```
{
```

```
    return null;
```

```
}
```

```
var failoverClusterIds = failoverClusterRelationships
```

```
.Select(r => r.ChildConfigurationItem.SysId)
```

```
.Distinct()
```

```
.ToList();
```

```
_logger.Information("Getting Windows Failover Cluster Applications");
```

```
var failoverClusterApplications = GetWindowsFailoverApplications(failoverClusterIds, cancellationToken);
```

```
_logger.Information($"Windows Failover Cluster Applications Found: {failoverClusterApplications.Count}");
```

```
if (cancellationToken.IsCancellationRequested)
```

```
{
```

```
    return null;
```

```
}
```

```
_logger.Information("Getting SQL Sever Instances");
```

```
var sqlServerInstances = GetSqlServerInstances(relationships, cancellationToken);
```

```
_logger.Information($"SQL Sever Instances Found: {sqlServerInstances.Count}");
```

```
// Map the values in each collection to the properties on each server.
```

```
servers.ForEach(
```

```
    s =>
```

```
{
```

```
    s.FileSystems = fileSystems
```

```
        .Where(f => f.ComputerSysId == s.SysId)
```

```
        .OrderBy(f => string.IsNullOrEmpty(f.Label) ? f.Name : f.Label)
```

```
        .ToList();
```

```
    s.Relationships = relationships
```

```
        .Where(r => r.ParentSysId == s.SysId || r.ChildSysId == s.SysId)
```

```
        .ToList();
```

```
    s.SqlServerInstances = s.Relationships
```

```
        .Where(r => string.Equals(r.ParentConfigurationItem.SysClassName, "cmdb_ci_db_mssql_instance",
```

```
StringComparison.OrdinalIgnoreCase))
```

```
        .Select(r => sqlServerInstances.FirstOrDefault(ss => string.Equals(ss.SysId, r.ParentSysId,
```

```
StringComparison.OrdinalIgnoreCase)))
```

```
.ToList();
```

```
s.SoftwareInstances = softwareInstances
```

```
.Where(si => si.InstalledOnServerSysId == s.SysId)
```

```
.GroupBy(si => new { si.NormalizedDisplayName, si.NormalizedPublisher, si.NormalizedVersion }, (key, si) =>
```

```
si.First())
```

```
.OrderBy(si => si.NormalizedDisplayName)
```

```
.ToList();
```

```
s.Location = locations.FirstOrDefault(l => l.SysId == s.LocationSysId);
```

```
string failoverClusterNodeSysId = s.Relationships.Where(IsFailoverCluster)
```

```
.Select(r => r.ParentConfigurationItem.SysId)
```

```
.FirstOrDefault();
```

```
if (string.IsNullOrEmpty(failoverClusterNodeSysId) == false)
```

```
{
```

```
// The failover cluster is the child of the relationship.
```

```
s.WindowsServerFailoverCluster = failoverClusterRelationships.Where(r => r.ParentSysId ==
```

```
failoverClusterNodeSysId).Select(r => r.ChildConfigurationItem).FirstOrDefault();
```

```
if (s.WindowsServerFailoverCluster != null)
```

```
{
```

```
s.WindowsFailoverApps = failoverClusterApplications.Where(a => a.ClusterSysId ==
```

```
s.WindowsServerFailoverCluster.SysId).ToList();
```

```
    }  
    }  
});  
}
```

```
_logger.Information($"End GetServers() {servers.Count} servers found.");
```

```
return servers;
```

```
}
```

```
private List<ServiceNowSqlServerInstanceModel> GetSqlServerInstances(List<ServiceNowRelationshipModel>  
relationships, CancellationToken cancellationToken)
```

```
{
```

```
    var sqlInstanceIds = relationships
```

```
        .Where(r => string.Equals(r.ParentConfigurationItem.SysClassName, "cmdb_ci_db_mssql_instance",
```

```
StringComparison.OrdinalIgnoreCase))
```

```
        .Select(ci => ci.ParentConfigurationItem.SysId).ToList();
```

```
    var sqlServerInstances = new List<ServiceNowSqlServerInstanceModel>();
```

```
    if (sqlInstanceIds.Count > 0)
```

```
{
```

```
    var sb = new StringBuilder()
```

```
        .Append(TableApiBaseUrl)
```

```
        .Append("/cmdb_ci_db_mssql_instance")
```

```
        .Append("?sysparm_exclude_reference_link=true")
```

```

.Append("&sysparm_suppress_pagination_header=true")

.Append("&sysparm_limit=").Append(DefaultBatchSize)

.Append("&sysparm_offset={0}")

.Append("&sysparm_fields=name,sys_id,sys_class_name,tcp_port,instance_name")

.Append("&sysparm_query=sys_idIN{1}");

var results = GetResultsCollection<ServiceNowSqlServerInstanceModel>(sb.ToString(),
SubdivideAndConcatList(sqlInstanceIds, 100), cancellationToken);

sqlServerInstances.AddRange(results);

}

return sqlServerInstances;

}

private List<ServiceNowRelationshipModel> GetFailoverClusterRelationships(IReadOnlyCollection<string>
clusterNodeIds, CancellationToken cancellationToken)

{

return GetRelationships(clusterNodeIds, cancellationToken, "Cluster of::Cluster", false);

}

/// <summary>

/// Get the list of locations in Service Now.

/// </summary>

/// <returns>A collection of Service Now location models.</returns>

private List<ServiceNowLocationModel> GetLocations(CancellationToken cancellationToken)

{

```



```
var sb = new StringBuilder()

.Append(TableApiBaseUrl)

.Append("/cmn_location")

.Append("?sysparm_exclude_reference_link=true")

.Append("&sysparm_limit=").Append(DefaultBatchSize)

.Append("&sysparm_offset={0}")

.Append("&sysparm_fields=name,sys_id");
```

```
return GetResultsCollection<ServiceNowLocationModel>(sb.ToString(), cancellationToken);
```

```
}
```

```
private List<T> GetResultsCollection<T>(string restQuery, IEnumerable<string> parameterSet, CancellationToken
cancellationToken)
```

```
{
```

```
var completeResults = new List<T>();
```

```
foreach (var parameter in parameterSet)
```

```
{
```

```
var partialRestQuery = restQuery.Replace("{1}", parameter);
```

```
var partialResults = GetResultsCollection<T>(partialRestQuery, cancellationToken);
```

```
if (cancellationToken.IsCancellationRequested)
```

```
{
```

```
return null;
```

```
}
```

```

        completeResults.AddRange(partialResults);
    }

    return completeResults;
}

private List<T> GetResultsCollection<T>(string restQuery, CancellationToken cancellationToken)
{
    int? totalRecordCount;

    int recordsProcessed = 0;

    var completeResults = new List<T>();

    do
    {
        var targetedRestQuery = string.Format(restQuery, recordsProcessed);

        var partialResultsJson = ExecuteWebRequest(targetedRestQuery, "GET", out totalRecordCount);

        if (cancellationToken.IsCancellationRequested)
        {
            return null;
        }

        var partialResults = JsonConvert.DeserializeObject<ServiceNowApiResults<T>>(partialResultsJson);

        completeResults.AddRange(partialResults.Results);
    }
}

```

```

        recordsProcessed += partialResults.Results.Count;
    }

    while (recordsProcessed < totalRecordCount);

    return completeResults;
}

private string ExecuteWebRequest(string restQuery, string method, out int? totalCount)
{
    var request = (HttpWebRequest)WebRequest.Create(_url + restQuery);

    request.Accept = "application/json";

    request.ContentType = "application/json";

    request.Method = method;

    request.Credentials = new NetworkCredential(_userid, _password);

    request.CookieContainer = new CookieContainer();

    request.PreAuthenticate = true;

    var response = (HttpWebResponse)request.GetResponse();

    if (response.StatusCode != HttpStatusCode.OK)
    {
        throw new InvalidOperationException("The request returned with a status code other than 200. Status code: " +
response.StatusCode);
    }
}

```

```
string totalCountHeaderValue = response.Headers["X-Total-Count"];
```

```
totalCount = string.IsNullOrEmpty(totalCountHeaderValue) ? (int?)null : int.Parse(totalCountHeaderValue);
```

```
var responseStream = response.GetResponseStream();
```

```
if (responseStream == null)
```

```
{
```

```
    throw new NullReferenceException("The response stream was null. Unable to complete request. Request Url: " +  
request.RequestUri.ToString());
```

```
}
```

```
using (var responseReader = new StreamReader(responseStream))
```

```
{
```

```
    var responseString = responseReader.ReadToEnd();
```

```
    return responseString;
```

```
}
```

```
}
```

```
private static List<string> SubdivideAndConcatList(IEnumerable<string> strings, int bucketSize)
```

```
{
```

```
    return SubdivideList(strings, bucketSize).Select(ToDelimitedListString).ToList();
```

```
}
```

```
private static List<List<string>> SubdivideList(IEnumerable<string> strings, int bucketSize)
```

```

{

    var listGroups = new List<List<string>>();

    int valueNumber = 0;

    foreach (var value in strings)

    {

        if (valueNumber % bucketSize == 0)

        {

            var stringList = new List<string>();

            stringList.Add(value);

            listGroups.Add(stringList);

        }

        else

        {

            listGroups[listGroups.Count - 1].Add(value);

        }

        valueNumber++;

    }

    return listGroups;

}

private static string ToDelimitedListString(IEnumerable<string> list)

{

    var sb = new StringBuilder();

    bool firstElement = true;

```

```

foreach (var element in list)

{

    if (string.IsNullOrEmpty(element))

    {

        continue;

    }


    if (firstElement == false)

    {

        sb.Append(',');

    }


    sb.Append(element);

    firstElement = false;

}


return sb.Length > 0 ? sb.ToString() : null;

}

}

}

```

ServiceNowServerExtractor.cs (Monitoring.ETL.Domain\ServiceNowCmdb\Extract\ServiceNowServerExtractor.cs):

```

using System;

using System.Collections.Generic;

using System.Threading;

```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Extract
```

```
{
```

```
    public sealed class ServiceNowServerExtractor : IExtractor<ServiceNowServerModel>
```

```
    {
```

```
        public async Task<ResultSet<ServiceNowServerModel>> ExtractAllSinceLastExtractAsync(
```

```
            CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
        {
```

```
            var servers = await ExtractBetweenAsync(DateTime.MinValue, DateTime.MaxValue, cancellationToken, logger);
```

```
            if (cancellationToken.IsCancellationRequested)
```

```
            {
```

```
                return null;
```

```
            }
```

```
            var resultSet = new ResultSet<ServiceNowServerModel>();
```

```
            foreach (var server in servers)
```

```
            {
```

```
                resultSet.Results.Add(server);
```

```
            }
```

```
            return resultSet;
```

```
}
```

```
public Task<IEnumerable<ServiceNowServerModel>> ExtractBetweenAsync(DateTime startDate, DateTime  
endDate,
```

```
    Cancellation_token cancellationToken, IEtlProcessLogger logger)
```

```
{
```

```
    return Task.FromResult(GetServers(cancellationToken, logger));
```

```
}
```

```
private static IEnumerable<ServiceNowServerModel> GetServers(Cancellation_token cancellationToken,  
IEtlProcessLogger logger)
```

```
{
```

```
    try
```

```
    {
```

```
        var client = new ServiceNowRestApiClient(logger);
```

```
        return client.GetServers(cancellationToken);
```

```
    }
```

```
    catch (Exception e)
```

```
    {
```

```
        logger.Information("There was an error calling GetServers(). " + e.Message);
```

```
        throw;
```

```
    }
```

```
}
```

```
}
```

```
}
```


ServiceNowServerModel.cs (Monitoring.ETL.Domain\ServiceNowCmdb\Extract\ServiceNowServerModel.cs):

```
using System.Collections.Generic;
```

```
using ETL.Process;
```

```
using Newtonsoft.Json;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Extract
```

```
{
```

```
    /// <summary>
```

```
    /// Model for the Service Now Server entity.
```

```
    /// </summary>
```

```
    public class ServiceNowServerModel : ServiceNowConfigurationItemModel, IExtractModel
```

```
    {
```

```
        [JsonProperty("os")]
```

```
        public string Os { get; set; }
```

```
        [JsonProperty("os_version")]
```

```
        public string OsVersion { get; set; }
```

```
        [JsonProperty("os_service_pack")]
```

```
        public string OsServicePack { get; set; }
```

```
        [JsonProperty("ram")]
```

```
        public int? Ram { get; set; }
```

```
        [JsonProperty("cpu_speed")]
```

```
        public double? CpuSpeed { get; set; }
```

```
[JsonProperty("cpu_count")]
```

```
public int? CpuCount { get; set; }
```

```
[JsonProperty("cpu_core_count")]
```

```
public int? CpuCoreCount { get; set; }
```

```
[JsonProperty("cpu_core_thread")]
```

```
public int? CpuCoreThreadCount { get; set; }
```

```
[JsonProperty("fqdn")]
```

```
public string Fqdn { get; set; }
```

```
[JsonProperty("virtual")]
```

```
public bool Virtual { get; set; }
```

```
[JsonProperty("is_failover_cluster_member")]
```

```
public bool IsFailoverClusterMember => WindowsServerFailoverCluster != null;
```

```
[JsonProperty("total_physical_core_count")]
```

```
public int? TotalPhysicalCoreCount => CpuCount * CpuCoreCount;
```

```
[JsonProperty("total_logical_core_count")]
```

```
public int? TotalLogicalCoreCount => CpuCount * CpuCoreCount * (CpuCoreThreadCount ?? 1);
```

```
[JsonProperty("hyperthreaded")]
```

```
public bool? Hyperthreaded => CpuCoreThreadCount.HasValue ? CpuCoreThreadCount > 1 : (bool?)null;
```

```
public ServiceNowConfigurationItemModel WindowsServerFailoverCluster { get; set; }
```

```
[JsonProperty("os_name_and_version")]
```

```
public string OsNameAndVersionValue
```

```
{
```

```
    get
```

```
    {
```

```
        string value = Os;
```

```
        if (string.IsNullOrEmpty(OsVersion) == false)
```

```
        {
```

```
            value = $"{value} ({OsVersion})";
```

```
        }
```

```
        if (string.IsNullOrEmpty(OsServicePack) == false)
```

```
        {
```

```
            value = $"{value} {OsServicePack}";
```

```
        }
```

```
        return value;
```

```
    }
```

```
}
```

```
[JsonProperty("location")]
```

```
public string LocationSysId { get; set; }
```

```
public ServiceNowLocationModel Location { get; set; }
```

```
public List<ServiceNowFilesystemModel> Filesystems { get; set; }
```

```
public List<ServiceNowRelationshipModel> Relationships { get; set; }
```

```
public List<ServiceNowSoftwareInstallModel> SoftwareInstances { get; set; }
```

```
public List<ServiceNowWindowsFailoverAppModel> WindowsFailoverApps { get; set; }
```

```
public List<ServiceNowSqlServerInstanceModel> SqlServerInstances { get; set; }
```

```
}
```

```
}
```

ServiceNowSoftwareInstallModel.cs

(Monitoring.ETL.Domain\ServiceNowCmdb\Extract\ServiceNowSoftwareInstallModel.cs):

```
using Newtonsoft.Json;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Extract
```

```
{
```

```
public sealed class ServiceNowSoftwareInstallModel
```

```
{
```

```
[JsonProperty("sys_id")]
```

```
public string SysId { get; set; }
```

```
[JsonProperty("installed_on")]
```

```
public string InstalledOnServerSysId { get; set; }
```

```
[JsonProperty("normalized_display_name")]
```

```
public string NormalizedDisplayName { get; set; }
```

```
[JsonProperty("normalized_version")]
```

```
public string NormalizedVersion { get; set; }
```

```
[JsonProperty("normalized_publisher")]
```

```
public string NormalizedPublisher { get; set; }
```

```
}
```

```
}
```

ServiceNowSqlServerInstanceModel.cs

(Monitoring.ETL.Domain\ServiceNowCmdb\Extract\ServiceNowSqlServerInstanceModel.cs):

```
using Newtonsoft.Json;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Extract
```

```
{
```

```
public sealed class ServiceNowSqlServerInstanceModel : ServiceNowConfigurationItemModel
```

```
{
```

```
[JsonProperty("instance_name")]
```

```
public string InstanceName { get; set; }
```

```
[JsonProperty("tcp_port")]

public string Port { get; set; }

}

}
```

ServiceNowWindowsFailoverAppModel.cs

(Monitoring.ETL.Domain\ServiceNowCmdb\Extract\ServiceNowWindowsFailoverAppModel.cs):

```
using Newtonsoft.Json;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Extract
```

```
{

    public sealed class ServiceNowWindowsFailoverAppModel : ServiceNowConfigurationItemModel

    {

        [JsonProperty("u_cluster_node_running")]

        public string RunningOnClusterNodeSysId { get; set; }


        [JsonProperty("u_cluster")]

        public string ClusterSysId { get; set; }

    }

}
```

PlexFilesystemModel.cs (Monitoring.ETL.Domain\ServiceNowCmdb\Load\PlexFilesystemModel.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Load
```

```
{
```

```
public sealed class PlexFilesystemModel

{

    public string MountPoint { get; set; }

    public string FileSystem { get; set; }

    public Guid SysId { get; set; }

    public string Name { get; set; }

    public long FreeSpaceBytes { get; set; }

    public long SizeBytes { get; set; }

    public string Label { get; set; }

    public string Lun { get; set; }

    public string Device { get; set; }

    public string MediaType { get; set; }

    public string ProvidedBy { get; set; }

    public Guid ServerSysId { get; set; }

}

}
```

PlexLocationModel.cs (Monitoring.ETL.Domain\ServiceNowCmdb\Load\PlexLocationModel.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Load
```

```
{

    public sealed class PlexLocationModel

    {

        public string Name { get; set; }

        public Guid SysId { get; set; }

    }

}
```

```
        public Guid ServerSysId { get; set; }

    }

}
```

PlexServerLoader.cs (Monitoring.ETL.Domain\ServiceNowCmdb\Load\PlexServerLoader.cs):

```
using System;

using System.Collections.Generic;

using System.Data;

using System.IO;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using System.Xml;

using System.Xml.Serialization;

using ETL.Process;

using Libraries.Common.Exceptions;

using Monitoring.ETL.Domain.Warehouse;

using Monitoring.ETL.Process;

using Plex.Infrastructure.ConfigSystems;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Load

{

    public sealed class PlexServerLoader : ILoader<PlexServerModel>

    {

        private readonly IWarehouseServerProvider _warehouseServerProvider;
```



```

public PlexServerLoader()

    : this(new WarehouseServerProvider())

{

}


internal PlexServerLoader(IWarehouseServerProvider warehouseServerProvider)

{

    _warehouseServerProvider = warehouseServerProvider ?? throw new

ArgumentNullException(nameof(warehouseServerProvider));

}


public Task<bool> LoadAsync(IEnumerable<PlexServerModel> transformed, CancellationToken cancellationToken,

IEtlProcessLogger logger)

{

    if (cancellationToken.IsCancellationRequested)

    {

        return Task.FromResult(true);

    }


    var targetServerName = _warehouseServerProvider.GetWarehouseServerName();


    var connectionString = $"data source={targetServerName};initial catalog=Performance;integrated security=True";


    var transformedArray = transformed.ToArray();


    var allFileSystems = new List<PlexFilesystemModel>();

```

```
var allSoftwareInstalls = new List<PlexSoftwareInstallModel>();

var allSqlServerInstances = new List<PlexSqlServerInstanceModel>();

var allLocations = new List<PlexLocationModel>();

var allWindowsServerFailoverApplications = new List<PlexWindowsServerFailoverAppModel>();


foreach (var server in transformedArray)

{

    if (server.Filesystems?.Count > 0)

    {

        allFileSystems.AddRange(server.Filesystems);

    }


    if (server.SoftwareInstances?.Count > 0)

    {

        allSoftwareInstalls.AddRange(server.SoftwareInstances);

    }


    if (server.SqlServerInstances?.Count > 0)

    {

        allSqlServerInstances.AddRange(server.SqlServerInstances);

    }


    if (server.Location != null)

    {

        allLocations.Add(server.Location);

    }

}
```

```

if (server.WindowsServerFailoverApps?.Count > 0)
{
    allWindowsServerFailoverApplications.AddRange(server.WindowsServerFailoverApps);
}
}

var serverXml = XmlSerializeCollection(transformedArray);

var fileSystemXml = XmlSerializeCollection(allFileSystems.ToArray());

var softwareInstallXml = XmlSerializeCollection(allSoftwareInstalls.ToArray());

var sqlServerInstanceXml = XmlSerializeCollection(allSqlServerInstances.ToArray());

var locationXml = XmlSerializeCollection(allLocations.ToArray());

var windowsServerFailoverApplicationXml =
XmlSerializeCollection(allWindowsServerFailoverApplications.ToArray());

using (var con = new System.Data.SqlClient.SqlConnection(connectionString))
{
    con.Open();

    using (var cmd = new System.Data.SqlClient.SqlCommand("Performance.dbo.CMDB_Server_Sync", con))
    {
        AddXmlParameter(cmd, "@Server_Xml", serverXml);

        AddXmlParameter(cmd, "@Filesystem_Xml", fileSystemXml);

        AddXmlParameter(cmd, "@Software_Install_Xml", softwareInstallXml);

        AddXmlParameter(cmd, "@Sql_Server_Instance_Xml", sqlServerInstanceXml);

        AddXmlParameter(cmd, "@Location_Xml", locationXml);
    }
}

```

```
AddXmlParameter(cmd, "@Windows_Server_Failover_Application_Xml", windowsServerFailoverApplicationXml);
```

```
cmd.CommandTimeout = 300;
```

```
cmd.CommandType = CommandType.StoredProcedure;
```

```
cmd.ExecuteNonQuery();
```

```
}
```

```
}
```

```
return Task.FromResult(true);
```

```
}
```

```
private static void AddXmlParameter(System.Data.SqlClient.SqlCommand cmd, string parameterName, string  
parameterValue)
```

```
{
```

```
var parameter = cmd.CreateParameter();
```

```
parameter.SqlDbType = SqlDbType.Xml;
```

```
parameter.Value = parameterValue;
```

```
parameter.Size = parameterValue.Length;
```

```
parameter.ParameterName = parameterName;
```

```
cmd.Parameters.Add(parameter);
```

```
}
```

```
private static string XmlSerializeCollection<T>(T[] collection)
```

```
{
```

```
var xmlSerializer = new XmlSerializer(typeof(T[]));
```

```

using (var ms = new MemoryStream())

{
    using (var writer = XmlWriter.Create(ms, new XmlWriterSettings { Encoding = System.Text.Encoding.UTF8,
OmitXmlDeclaration = true }))

    {
        xmlSerializer.Serialize(writer, collection);

        var unicodeBytes = ms.ToArray();

        return System.Text.Encoding.UTF8.GetString(unicodeBytes);
    }
}
}
}
}
}
}
}

```

PlexServerModel.cs (Monitoring.ETL.Domain\ServiceNowCmdb\Load\PlexServerModel.cs):

```

using System;

using System.Collections.Generic;

using System.Xml.Serialization;

using ETL.Process;

namespace Monitoring.ETL.Domain.ServiceNowCmdb.Load
{
    /// <summary>

    /// Model for a transformed ServiceNow server.

```

/// </summary>

public sealed class PlexServerModel : ILoadModel

{

public string ServerName { get; set; }

public Guid SysId { get; set; }

public string SysClassName { get; set; }

public string Os { get; set; }

public string OsVersion { get; set; }

public string OsServicePack { get; set; }

public int? RamMb { get; set; }

public decimal? CpuSpeedMhz { get; set; }

public int? CpuCount { get; set; }

public int? CpuCoreCount { get; set; }

public int? CpuCoreThreadCount { get; set; }

public string FullyQualifiedDomainName { get; set; }

public PlexLocationModel Location { get; set; }

public PlexServerModel VirtualMachineHost { get; set; }

public string WindowsServerFailoverCluster { get; set; }

[XmlIgnore]

public List<PlexFilesystemModel> Filesystems { get; set; }

[XmlIgnore]

public List<PlexSoftwareInstallModel> SoftwareInstances { get; set; }

[XmlIgnore]

```
public List<PlexWindowsServerFailoverAppModel> WindowsServerFailoverApps { get; set; }
```

```
[XmlIgnore]
```

```
public List<PlexSqlServerInstanceModel> SqlServerInstances { get; set; }
```

```
}
```

```
}
```

PlexSoftwareInstallModel.cs (Monitoring.ETL.Domain\ServiceNowCmdb\Load\PlexSoftwareInstallModel.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Load
```

```
{
```

```
public sealed class PlexSoftwareInstallModel
```

```
{
```

```
public string Name { get; set; }
```

```
public string Version { get; set; }
```

```
public string Publisher { get; set; }
```

```
public Guid SysId { get; set; }
```

```
public Guid ServerSysId { get; set; }
```

```
}
```

```
}
```

PlexSqlServerInstanceModel.cs (Monitoring.ETL.Domain\ServiceNowCmdb\Load\PlexSqlServerInstanceModel.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.ServiceNowCmdb.Load
```

```

{

    public sealed class PlexSqlServerInstanceModel

    {

        public Guid SysId { get; set; }

        public string SysClassName { get; set; }

        public string Name { get; set; }

        public int? Port { get; set; }

        public Guid ServerSysId { get; set; }

    }

}

```

PlexWindowsServerFailoverAppModel.cs

(Monitoring.ETL.Domain\ServiceNowCmdb\Load\PlexWindowsServerFailoverAppModel.cs):

```

using System;

```

```

namespace Monitoring.ETL.Domain.ServiceNowCmdb.Load

```

```

{

    public sealed class PlexWindowsServerFailoverAppModel

    {

        public string Name { get; set; }

        public Guid SysId { get; set; }

        public string SysClassName { get; set; }

        public Guid ServerSysId { get; set; }

    }

}

```


ServiceNowServerExtractionDelayFactory.cs

(Monitoring.ETL.Domain\ServiceNowCmdb\ServiceNowServerExtractionDelayFactory.cs):

using System;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Domain.ServiceNowCmdb.Extract;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.ServiceNowCmdb

{

public class ServiceNowServerExtractionDelayFactory : IExtractionDelayFactory<ServiceNowServerModel>

{

public async Task Create(ResultSet<ServiceNowServerModel> results, CancellationToken cancellationToken,
IEtlProcessLogger logger)

{

// Tomorrow at 3:30AM.

DateTime nextRunTime = DateTime.Today.AddDays(1).AddHours(3.5);

logger.Information(\$"Sleeping until {nextRunTime}");

await Task.Delay(nextRunTime.Subtract(DateTime.Now), cancellationToken);

}

}

}

ServiceNowServerTransformer.cs

(Monitoring.ETL.Domain\ServiceNowCmdb\Transform\ServiceNowServerTransformer.cs):

```

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Domain.ServiceNowCmdB.Extract;

using Monitoring.ETL.Domain.ServiceNowCmdB.Load;

using Monitoring.ETL.Process;


namespace Monitoring.ETL.Domain.ServiceNowCmdB.Transform
{
    public sealed class ServiceNowServerTransformer : ITransformer<ServiceNowServerModel, PlexServerModel>
    {
        public Task<IEnumerable<PlexServerModel>> TransformAsync(IEnumerable<ServiceNowServerModel> extracted,
CancellationTok
en cancellationToken, IEtlProcessLogger logger)
        {
            if (cancellationToken.IsCancellationRequested)
            {
                return Task.FromResult<IEnumerable<PlexServerModel>>(null);
            }

            return Task.FromResult(
                extracted.Select(
                    s =>
                        new PlexServerModel

```

```

{

    ServerName = s.Name,

    CpuCount = s.CpuCount,

    CpuCoreCount = s.CpuCoreCount,

    CpuCoreThreadCount = s.CpuCoreThreadCount,

    CpuSpeedMhz = s.CpuSpeed.HasValue ? (decimal?)s.CpuSpeed : null,

    FullyQualifiedDomainName = s.Fqdn,


    Location = s.Location == null

        ? null

        : new PlexLocationModel

    {

        Name = s.Location.Name,

        SysId = Guid.Parse(s.Location.SysId),

        ServerSysId = Guid.Parse(s.SysId)

    },


    VirtualMachineHost = s.Relationships

        ?.Where(r => string.Equals(r.RelationshipType.Name, "Virtualized by::Virtualizes",

StringComparison.OrdinalIgnoreCase))

        .Where(r => string.Equals(r.ChildConfigurationItem.Name, s.Name, StringComparison.OrdinalIgnoreCase) ==

false)

        .Select(

            r => new PlexServerModel

            {

                ServerName = r.ChildConfigurationItem.Name,

```

```
SysId = Guid.Parse(r.ChildConfigurationItem.SysId),  
  
SysClassName = r.ChildConfigurationItem.SysClassName  
  
}).FirstOrDefault(),
```

```
WindowsServerFailoverCluster = s.WindowsServerFailoverCluster?.Name,
```

```
WindowsServerFailoverApps = s.WindowsFailoverApps
```

```
?.Select(  
  
a => new PlexWindowsServerFailoverAppModel  
  
{  
  
    Name = a.Name,  
  
    SysClassName = a.SysClassName,  
  
    SysId = Guid.Parse(a.SysId),  
  
    ServerSysId = Guid.Parse(s.SysId)  
  
}).ToList(),
```

```
Filesystems = s.Filesystems
```

```
?.Select(  
  
fs => new PlexFilesystemModel  
  
{  
  
    Device = fs.Device,  
  
    FileSystem = fs.FileSystem,  
  
    FreeSpaceBytes = fs.FreeSpaceBytes ?? 0,  
  
    Label = fs.Label,  
  
    Lun = fs.Lun,  
  
    MediaType = fs.MediaType,
```

```
MountPoint = fs.MountPoint,  
  
Name = fs.Name,  
  
ProvidedBy = fs.ProvidedBy?.Name,  
  
SizeBytes = fs.SizeBytes ?? 0,  
  
SysId = Guid.Parse(fs.SysId),  
  
ServerSysId = Guid.Parse(s.SysId)  
  
}).ToList(),
```

```
SoftwareInstances = s.SoftwareInstances
```

```
?.Select(  
  
    si => new PlexSoftwareInstallModel  
  
    {  
  
        Name = si.NormalizedDisplayName,  
  
        Publisher = si.NormalizedPublisher,  
  
        Version = si.NormalizedVersion,  
  
        SysId = Guid.Parse(si.SysId),  
  
        ServerSysId = Guid.Parse(s.SysId)  
  
    }).ToList(),
```

```
Os = s.Os,  
  
OsServicePack = s.OsServicePack,  
  
OsVersion = s.OsVersion,  
  
RamMb = s.Ram,  
  
SysClassName = s.SysClassName,  
  
SysId = Guid.Parse(s.SysId),
```

```
SqlServerInstances = s.SqlServerInstances
```

```
?.Select(ss => new PlexSqlServerInstanceModel
```

```
{
```

```
    Name = ss.InstanceName,
```

```
    Port = int.TryParse(ss.Port, out var result) && result > 0 ? result : (int?)null,
```

```
    SysId = Guid.Parse(ss.SysId),
```

```
    SysClassName = ss.SysClassName,
```

```
    ServerSysId = Guid.Parse(s.SysId)
```

```
}).ToList()
```

```
    ));
```

```
}
```

```
}
```

```
}
```

ServiceReference.cs (Monitoring.ETL.Domain\ServiceReference.cs):

```
namespace Monitoring.ETL.Domain
```

```
{
```

```
    public class ServiceReference
```

```
    {
```

```
        // Do not delete this class.
```

```
        // This is used as a soft reference from
```

```
        // the service to load this dll into memory
```

```
        // at runtime.
```

```
    }
```

```
}
```

ServiceToolConnectionRepository.cs (Monitoring.ETL.Domain\ServiceToolConnectionRepository.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Data;
```

```
using System.Data.SqlClient;
```

```
namespace Monitoring.ETL.Domain
```

```
{
```

```
    internal class ServiceToolConnectionRepository
```

```
    {
```

```
        private const string TrustedSqlServerConnectionStringTemplate = "data source={0};initial
```

```
catalog=Plexus_Control;integrated security=True;";
```

```
        private readonly IWarehouseServerProvider _warehouseServerProvider = new WarehouseServerProvider();
```

```
        public IDictionary<string, SqlServerModel> GetServersToProcess(string serviceToolName)
```

```
        {
```

```
            var servers = new Dictionary<string, SqlServerModel>();
```

```
            var warehouseServer = _warehouseServerProvider.GetWarehouseServerName();
```

```
            try
```

```
            {
```

```
                using (var con = new SqlConnection(string.Format(TrustedSqlServerConnectionStringTemplate,
```

```
warehouseServer)))
```

```
                {
```

```
con.Open();

using (var cmd = new SqlCommand("Performance.dbo.Plex_Services_Sql_Server_Get", con))
{
    cmd.CommandType = CommandType.StoredProcedure;

    cmd.Parameters.Add(
        new SqlParameter
        {
            ParameterName = "@Plex_Service_Tool_Name",
            SqlDbType = SqlDbType.VarChar,
            Value = serviceToolName
        });

    using (var reader = cmd.ExecuteReader())
    {
        if (reader.HasRows)
        {
            var serviceNameOrdinal = reader.GetOrdinal("Plex_Service_Name");
            var serviceCommonNameOrdinal = reader.GetOrdinal("Service_Common_Name");
            var serverNameOrdinal = reader.GetOrdinal("Plex_Server_Name");
            var portNumberOrdinal = reader.GetOrdinal("Port_Number");

            while (reader.Read())
            {
                var serviceName = reader.GetString(serviceNameOrdinal);
```



```
var serviceCommonName = reader.GetString(serviceCommonNameOrdinal);

var serverName = reader.GetString(serverNameOrdinal);

var portNumber = reader.GetInt32(portNumberOrdinal);


servers.Add(

    string.IsNullOrEmpty(serviceCommonName) ? serviceName : serviceCommonName,

    new SqlServerModel

    {

        InstanceName = serviceName,

        Port = portNumber,

        MachineName = serverName

    });

}

}

}

}

}

}

}

catch (Exception e)

{

    e.Data.Add("X-Server", warehouseServer);

    throw;

}

return servers;

}
```

```
internal class SqlServerModel

{

    public string InstanceName { get; set; }

    public string MachineName { get; set; }

    public int Port { get; set; }

}

}
```

IJsonExtractModel.cs (Monitoring.ETL.Domain\SqlJson\IJsonExtractModel.cs):

```
namespace Monitoring.ETL.Domain.SqlJson

{

    public interface IJsonExtractModel

    {

        /// <summary>

        /// The JSON object which contains the data to load

        /// </summary>

        string JsonMessage { get; set; }

    }

}
```

Repository.cs (Monitoring.ETL.Domain\SqlJson\Repository.cs):

```
using ETL.Process;

using Monitoring.ETL.Domain.ElasticSearch;
```

```
using Monitoring.ETL.Process;
```

```
using Newtonsoft.Json;
```

```
using Newtonsoft.Json.Linq;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Data;
```

```
using System.Data.SqlClient;
```

```
using System.Linq;
```

```
using System.Text;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using System.Xml.Linq;
```

```
namespace Monitoring.ETL.Domain.SqlJson
```

```
{  
  
    public class Repository<T> where T : IJsonExtractModel, new()  
  
    {  
  
        private readonly int _batchSize;  
  
        private readonly IEnumerable<string> _consulIds;  
  
        private readonly string _databaseName;  
  
        private readonly string _loadName;  
  
        private readonly string _maxIdColumn;  
  
        private readonly LoadStateRepository _state = new LoadStateRepository();  
  
        private readonly bool _useState;
```

```
private string _queryTemplate;
```

```
public Repository(IEnumerable<string> consullDs, string databaseName, string queryTemplate, int batchSize =  
10000,
```

```
string maxIdColumn = null, string stateLoadName = null)
```

```
{
```

```
    _consullDs = consullDs;
```

```
    _databaseName = databaseName;
```

```
    _queryTemplate = queryTemplate;
```

```
    _batchSize = batchSize;
```

```
    _maxIdColumn = maxIdColumn;
```

```
    _loadName = stateLoadName;
```

```
    _useState = !string.IsNullOrEmpty(stateLoadName);
```

```
}
```

```
public async Task<ResultSet<T>> ExtractAllSinceLastExtractAsync(
```

```
    CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
{
```

```
    var results = new ResultSet<T>();
```

```
    var loadStateUpdates = new Dictionary<string, long>();
```

```
    try
```

```
    {
```

```
        foreach (var consullId in _consullDs)
```

```
        {
```

```
            logger.Information($"Processing ConsullId: {consullId} ...");
```

```
var loadName = $"_{loadName}:{consulId}";
```

```
//TODO: Expand support to work with both bigint and datetime checkpoint values when the need arises.
```

```
long? checkpointId = null;
```

```
if (_useState)
```

```
{
```

```
    checkpointId = _state.GetBigIntKeyFor(loadName);
```

```
}
```

```
try
```

```
{
```

```
    var records = await GetNewRecords(consulId, checkpointId);
```

```
    logger.Information($"Success - {records?.Count} record(s) found.");
```

```
    if (records?.Count > 0)
```

```
    {
```

```
        if (_useState)
```

```
        {
```

```
            loadStateUpdates.Add(loadName, GetMaxIdFromRecords(records));
```

```
        }
```

```
        foreach (var record in records)
```

```
        {
```

```
            results.Results.Add(record);
```

```
        }
```

```
    }
```

```
}
```

```

catch (Exception e)

{

    if (IsNetworkConnectionException(e))

    {

        logger.Information("Instance currently unavailable. Skipping extraction for this execution.");

    }

    else

    {

        logger.Information("Failed!");

        e.Data.Add("X-ConsulId", consulId);

        results.Exceptions.Add(e);

    }

}

// If cancellation is requested, abort and return an empty result set.

// Note that we are NOT updating the load states otherwise we would have data loss.

if (cancellationToken.IsCancellationRequested)

{

    logger.Information("Cancellation requested - discarding results and exiting.");

    return new ResultSet<T>();

}

}

// After all servers have been processed, persist the keys to the load states.

if (_useState)

{

```

```

        foreach (var loadStateUpdate in loadStateUpdates)
        {
            await _state.SetStateAsync(loadStateUpdate.Key, loadStateUpdate.Value);
        }
    }

    return results;
}

catch (Exception ex)
{
    results.Exceptions.Add(ex);

    return results;
}
}

public async Task<IEnumerable<T>> ExtractBetweenAsync(DateTime startDate,
    DateTime endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)
{
    throw new NotImplementedException();
}

/// <summary>
/// Sets the query template
/// </summary>
/// <param name="queryTemplate"></param>
public void SetQueryTemplate(string queryTemplate)

```

```
{  
  
    _queryTemplate = queryTemplate;  
  
}
```

```
private static bool IsNetworkConnectionException(Exception ex)
```

```
{  
  
    if (ex is SqlException sqlException)  
  
    {  
  
        for (var i = 0; i < sqlException.Errors.Count; i++)  
  
        {  
  
            if (sqlException.Errors[i].Number == 1225)  
  
            {  
  
                return true;  
  
            }  
  
        }  
  
    }  
  
}
```

```
    return false;
```

```
}
```

```
private long GetMaxIdFromRecords(IEnumerable<T> records)
```

```
{  
  
    var lastRecord = records.LastOrDefault();  
  
    if (lastRecord == null)  
  
    {  
  
        return 0;
```



```
}
```

```
var json = JObject.Parse(lastRecord.JsonMessage);
```

```
return long.Parse((string)json.SelectToken(_maxIdColumn) ?? "0");
```

```
}
```

```
private async Task<List<T>> GetNewRecords(string consuld, long? checkpointId = null)
```

```
{
```

```
var connectionString = Helpers.Util.GetSqlConnectionStringFromConsuld(consuld, _databaseName);
```

```
var results = new List<T>();
```

```
if (string.IsNullOrEmpty(connectionString))
```

```
    throw new Exception("Invalid connection string");
```

```
using (var connection = new SqlConnection(connectionString))
```

```
{
```

```
    connection.Open();
```

```
var query = _queryTemplate
```

```
    .Replace("{batchSize}", _batchSize.ToString())
```

```
    .Replace("{consuld}", consuld)
```

```
    .Replace("{checkpointId}", (checkpointId ?? 0).ToString());
```

```
using (var command = new SqlCommand(query, connection))
```

```
{
```

```
    command.CommandType = CommandType.Text;
```

```
var reader = await command.ExecuteReaderAsync();
```

```

var json = new StringBuilder();

while (await reader.ReadAsync())

{
    json.Append(reader.GetValue(0));
}

if (json.Length > 0)

{
    var records = JArray.Parse(json.ToString())

        .Select(x => x.ToString(Formatting.None)).ToList();

    foreach (var record in records)

    {
        results.Add(new T { JsonMessage = record });
    }
}

}

return results;

}

}

}

```

DelayFactory.cs (Monitoring.ETL.Domain\Template\RabbitMQExtractor\DelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.Template.RabbitMQExtractor
```

```
{
```

```
/// <summary>
```

```
/// Defines how often the the data load runs. If the total number of records loaded is beyond the
```

```
/// MaxThresholdForDelay, then no delay occurs and the load process will immediately be executed again.
```

```
///
```

```
/// If the number of records is less than the Threshold, then the process will sleep for DelayTimeSpan amount of time
```

```
///
```

```
/// The data Extractor process should load data in chunks of records for a specific date range.
```

```
/// </summary>
```

```
public sealed class DelayFactory : ThresholdExtractionDelayFactory<RabbitMQ.Model.RabbitMQTemplateModel>
```

```
{
```

```
/// <summary>
```

```
/// Time to delay when the threshold count is not reached
```

```
/// </summary>
```

```
protected override TimeSpan DelayTimeSpan => new TimeSpan(0, 0, 30);
```

```
/// <summary>
```

```
/// number of records to reach before the delay is ignored.
```

```
/// </summary>
```

```
protected override int MaxThresholdForDelay => 5;
```

```
}
```

```
}
```

Extractor.cs (Monitoring.ETL.Domain\Template\RabbitMQExtractor\Extractor.cs):

using ETL.Process;

using Monitoring.ETL.Process;

using System;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;

namespace Monitoring.ETL.Domain.Template.RabbitMQExtractor

{

/// <summary>

/// Extras data from Rabbit MQ. You only need to change the RabbitMQTemplateModel to

/// your model name

/// </summary>

public sealed class Extractor : IExtractor<RabbitMQ.Model.RabbitMQTemplateModel>

{

private RabbitMQ.Template.Consumer _consumer;

public async Task<ResultSet<RabbitMQ.Model.RabbitMQTemplateModel>>

ExtractAllSinceLastExtractAsync(CancellationToken cancellationToken, IEtlProcessLogger logger)

{

_consumer = _consumer ?? new RabbitMQ.Template.Consumer(logger);

return await _consumer.ExtractAsync();

}

```

        public Task<IEnumerable<RabbitMQ.Model.RabbitMQTemplateModel>> ExtractBetweenAsync(DateTime
startDate, DateTime endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)
    {
        throw new NotImplementedException();
    }
}
}
}

```

TemplateWarehouseLoader.cs (Monitoring.ETL.Domain\Template\RabbitMQExtractor\TemplateWarehouseLoader.cs):

```

using Monitoring.ETL.Domain.Warehouse;

```

```

namespace Monitoring.ETL.Domain.Template.RabbitMQExtractor

```

```

{
    /// <summary>
    /// The loader load the Template_Worktable in the Meta_Warehouse executes a stored procedure on the
Meta_Warehouse table
    /// which takes data from your work table and loads any fact and dimension tables as required
    ///
    /// Change the string "Fact_Template" to the name of your spoc which is transferring data
    /// from your work table (in this case "Template_Worktable") to the various fact and dimension tables
    /// </summary>

```

```

    public class TemplateWarehouseLoader : Loader<Warehouse.Model.Template_Worktable>

```

```

    {
        public TemplateWarehouseLoader() : base("Fact_Template")
        {

```

```
    }  
}  
}
```

Transformer.cs (Monitoring.ETL.Domain\Template\RabbitMQExtractor\Transformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Template.RabbitMQExtractor
```

```
{
```

```
    /// <summary>
```

```
    /// Load the messages from RabbitMQ into the table Template_Worktable.
```

```
    /// </summary>
```

```
    public class Transformer : ITransformer<RabbitMQ.Model.RabbitMQTemplateModel,
```

```
    Warehouse.Model.Template_Worktable>
```

```
    {
```

```
        public async Task<IEnumerable<Warehouse.Model.Template_Worktable>>
```

```
        TransformAsync(IEnumerable<RabbitMQ.Model.RabbitMQTemplateModel> extracted, CancellationToken
```

```
        cancellationToken, IEtlProcessLogger logger)
```

```
        {
```

```
await Task.Yield();
```

```
return extracted.Select(pc => new Warehouse.Model.Template_Worktable
```

```
{
```

```
    JSON_Message = pc.JsonMessage
```

```
});
```

```
}
```

```
}
```

```
}
```

DelayFactory.cs (Monitoring.ETL.Domain\Template\RabbitMQLoader\DelayFactory.cs):

```
using System;
```

```
using System.Configuration;
```

```
namespace Monitoring.ETL.Domain.Template.RabbitMQLoader
```

```
{
```

```
    /// <summary>
```

```
    /// Creates delay rules on when the process can run. If the number of records isn't reached,
```

```
    /// then the processed is delayed for the amount of time set before executing again
```

```
    /// </summary>
```

```
public class DelayFactory : ThresholdExtractionDelayFactory<MyDataModel>
```

```
{
```

```
    /// <summary>
```

```
    /// Ten minute wait when threshold is not reached
```

```
    /// </summary>
```

```
private TimeSpan _delayTimespan = new TimeSpan(0, 10, 0);
```

```
/// <summary>
```

```
/// Number of records to reach before the delay is ignored
```

```
/// </summary>
```

```
private int _maxThresholdForDelay = 10;
```

```
protected override TimeSpan DelayTimeSpan => _delayTimespan;
```

```
protected override int MaxThresholdForDelay => _maxThresholdForDelay;
```

```
public DelayFactory()
```

```
{
```

```
    //You could also include more robust logic in setting the delay here.
```

```
}
```

```
}
```

```
}
```

Extractor.cs (Monitoring.ETL.Domain\Template\RabbitMQLoader\Extractor.cs):

```
using ETL.Process;
```

```
using Monitoring.Email.Models;
```

```
using Monitoring.Email.Services;
```

```
using Monitoring.ETL.Domain.ElasticSearch;
```

```
using Monitoring.ETL.Process;
```

```
using Newtonsoft.Json;
```

```
using Newtonsoft.Json.Linq;
```



```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Configuration;
```

```
using System.Linq;
```

```
using System.Net.Http;
```

```
using System.Net.Http.Headers;
```

```
using System.Text;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.Template.RabbitMQLoader
```

```
{
```

```
    /// <summary>
```

```
    /// Extract data from your data source in chunks of records, either by a date range, or a unique key range.
```

```
    /// The range is required so you don't keep reloading old data.
```

```
    ///
```

```
    /// The _loadStateRepository represents a row in the Load_State table which contains the last load status
```

```
    /// of your ETL service
```

```
    /// </summary>
```

```
    public class Extractor : IExtractor<MyDataModel>
```

```
    {
```

```
        /// <summary>
```

```
        /// Contains information about the last time this ETL was loaded. You can keep track of:
```

```
        ///
```

```
        /// Last_Load_Date (datetime)
```

```

/// Last_Load_Key (int)

/// Last_Load_Bigint_Key (bigint)

///

/// Your ExtractAllSinceLastExtractAsync function should look at the appropriate value and use

/// it to determine the next block of records to load.

///

/// After you get your block of records, you must update your _loadStateRepository with a new date, key, or
bigint_key to the last

/// value loaded.

/// </summary>

private readonly LoadStateRepository<MyDataModel> _loadStateRepository;

/// <summary>

/// Container for your data

/// </summary>

private ResultSet<MyDataModel> resultSet;

public Extractor()

{

    /// Get the current load state for the particular model. If there is more than 1 source

    /// of data where you have to keep track of the load state, that name/id can be set in the

    /// LoadStateRepository overload

_loadStateRepository = new LoadStateRepository<MyDataModel>();

}

public async Task<ResultSet<MyDataModel>> ExtractAllSinceLastExtractAsync(CancellationTok

```

cancellationToken, IEtlProcessLogger logger)

```
{  
  
    resultSet = new ResultSet<MyDataModel>();           // Create an empty result set of JSON Objects  
  
    var loadMyData = MyLoadMethod(_loadStateRepository.GetKey()); // load my data with whatever means  
necessary for your ETL  
  
    var maxUserId = loadMyData.Max(d => d.InternalUserNumber); // Find the maximum user ID that we are  
going to load  
  
    await _loadStateRepository.SetKey(maxUserId);       // Set that maximum value in the Load_State table  
so we don't reload old data the next time the process is run  
  
    // only return a resultSet if there were new records to load  
  
    if (loadMyData.Any())  
    {  
  
        var json = JsonConvert.SerializeObject(loadMyData); //serialize into json  
  
        resultSet.Results.Add(new MyDataModel { JsonMessage = json }); //add to the record set  
  
        return resultSet; // this recordset will be deemed as 1 "message" in rabbitMQ,  
        // even though there could be  
        // more than 1 user to add  
    }  
  
    else  
  
        return null;  
}  
  
/// <summary>  
  
/// This is your main function. This will load data from whatever data sources you have, and place them in a list of
```

your model.

/// Your extractor should recognize the _loadStateRepository values so you're not reloading the same data over and over. In this example

/// we have a list of users with an internal user number, and we are only loading those users we haven't yet loaded.

/// </summary>

/// <returns></returns>

```
private List<MyUserData> MyLoadMethod(int lastUserNumberToLoad)
```

```
{
```

```
    ///Get all data that we haven't loaded yet
```

```
    var usersToAdd = MockUserData().Where(d => d.InternalUserNumber > lastUserNumberToLoad).ToList();
```

```
    return usersToAdd;
```

```
}
```

/// <summary>

/// A helper function which is mocking up some data for this example

/// </summary>

/// <returns></returns>

```
private List<MyUserData> MockUserData()
```

```
{
```

```
    var mylist = new List<MyUserData>();
```

```
    mylist.Add(new MyUserData { InternalUserNumber = 1, EmailAddress = "dwilson@plex.com", FirstName =  
"Dave", LastName = "Wilson" });
```

```
    mylist.Add(new MyUserData { InternalUserNumber = 2, EmailAddress = "ctaegan@plex.com", FirstName =  
"Charley", LastName = "Taegan" });
```

```
        mylist.Add(new MyUserData { InternalUserNumber = 3, EmailAddress = "vmerrilyn@plex.com", FirstName =
"Vivian", LastName = "Merrilyn" });

        mylist.Add(new MyUserData { InternalUserNumber = 4, EmailAddress = "jjayne@plex.com", FirstName =
"Jeanne", LastName = "Jayne" });

        mylist.Add(new MyUserData { InternalUserNumber = 5, EmailAddress = "gmichael@plex.com", FirstName =
"Gilbert", LastName = "Michael" });

        mylist.Add(new MyUserData { InternalUserNumber = 6, EmailAddress = "flevitt@plex.com", FirstName =
"Femie", LastName = "Levitt" });

        mylist.Add(new MyUserData { InternalUserNumber = 7, EmailAddress = "sthornton@plex.com", FirstName =
"Sterling", LastName = "Thornton" });

        mylist.Add(new MyUserData { InternalUserNumber = 8, EmailAddress = "dbutcher@plex.com", FirstName =
"Dustin", LastName = "Butcher" });

        mylist.Add(new MyUserData { InternalUserNumber = 9, EmailAddress = "rallsopp@plex.com", FirstName =
"Roxanna", LastName = "Allsopp" });

        mylist.Add(new MyUserData { InternalUserNumber = 10, EmailAddress = "cnye@plex.com", FirstName =
"Cyrus", LastName = "Nye" });

        return mylist;
    }
}
```

/// <summary>

/// This method is not yet used

/// </summary>

/// <param name="startDate"></param>

```

/// <param name="endDate"></param>

/// <param name="cancellationToken"></param>

/// <param name="logger"></param>

/// <returns></returns>

public Task<IEnumerable<MyDataModel>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
Cancellation token cancellationToken, IEtlProcessLogger logger)

{

    throw new NotImplementedException();

}

/// <summary>

/// A sample class of data from your data source to load

/// </summary>

public class MyUserData

{

    public int InternalUserNumber { get; set; }

    public string EmailAddress { get; set; }

    public string FirstName { get; set; }

    public string LastName { get; set; }

}

}

}

```

MyDataModel.cs (Monitoring.ETL.Domain\Template\RabbitMQLoader\MyDataModel.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.SqlJson;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Text;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.Template.RabbitMQLoader
```

```
{
```

```
    /// <summary>
```

```
    /// A model which holds the data which you're loading from your source. You will need to convert
```

```
    /// your data into a Json object which populates the JsonMessage property. This JSON object
```

```
    /// will then be loaded into RabbitMQ
```

```
    /// </summary>
```

```
    public class MyDataModel : IExtractModel, IJsonExtractModel, ILoadModel
```

```
    {
```

```
        /// <summary>
```

```
        /// The data to load, in a JSON Object
```

```
        /// </summary>
```

```
        public string JsonMessage { get; set; }
```

```
    }
```

```
}
```

TemplateRabbitMqLoader.cs (Monitoring.ETL.Domain\Template\RabbitMQLoader\TemplateRabbitMqLoader.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Text;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.Template.RabbitMQLoader
```

```
{
```

```
    /// <summary>
```

```
    /// Wrapper class
```

```
    /// </summary>
```

```
    public sealed class TemplateRabbitMqLoader : RabbitMQ.Loader<RabbitMQ.Model.RabbitMQTemplateModel>
```

```
    {
```

```
        public TemplateRabbitMqLoader() : base(new RabbitMQ.Template.Producer())
```

```
        {
```

```
        }
```

```
    }
```

```
}
```

```
Transformer.cs (Monitoring.ETL.Domain\Template\RabbitMQLoader\Transformer.cs):
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```



```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Template.RabbitMQLoader
```

```
{
```

```
    /// <summary>
```

```
    /// This function can be used to transform the data for whatever reason as required. Usually not used.
```

```
    /// </summary>
```

```
    public class Transformer : ITransformer<MyDataModel, RabbitMQ.Model.RabbitMQTemplateModel>
```

```
    {
```

```
        public async Task<IEnumerable<RabbitMQ.Model.RabbitMQTemplateModel>>
```

```
        TransformAsync(IEnumerable<MyDataModel> extracted, CancellationToken cancellationToken, IEtlProcessLogger  
        logger)
```

```
        {
```

```
            await Task.Yield();
```

```
            return extracted.Select(pc =>
```

```
                new RabbitMQ.Model.RabbitMQTemplateModel
```

```
                {
```

```
                    JsonMessage = pc.JsonMessage
```

```
                });
```

```
            }
```

```
        }
```

```
    }
```

```
readme.md (Monitoring.ETL.Domain\Template\readme.md):
```

ETL Template

This is a template ETL service. It lists the main parts of your ETL service which you will need for a functional ETL.

Overview

The monitoring ETL service has a defined flow for the data:

1. Create a RabbitMQ Queue to store messages. This is a one time execution handled by the ETL wrapper based on your Rabbit MQ Configuration settings in a specific class that you create.
2. Load data from your data source(s) and place into the queue. The benefit of the queue is that data can be loaded from a variety of sources before being processed for the final destination in the Meta_Warehouse table. The data in RabbitMQ is expected to be a JSON object
3. Extract the data from your RabbitMQ Queue into a SQL Work Table table
4. Execute your store procedure for reading the JSON data from the work table and load or match keys in various DIMENSION tables, and eventually load for final FACT table

Rabbit MQ Configuration

The first is to set up your RabbitMQ configuration. This can be found in the folder:

****RabbitMQ/Template****

These three classes named Configuration, Consumer, and Producer essentially define your RabbitMQ Name and sets up some internal piping

for the ETL wrapper to use. The Consumer and Producer can be simply copy/paste as is. The

Configuration class can be copied and then then **Queue** and **RoutingKeyMessageSourceComponent** properties can be set to configure your queue. These two properties values should be lower case.

The Queue is the queue name which will appear in RabbitMQ. Generally the naming convention will be:

monitoring.warehouse.[mynamehere]

Rabbit MQ Loader

These set of classes and method retrieve data from your data source

and load them into RabbitMQ as a JSON object for later processing by the RabbitMQExtractor

Rabbit MQ Extractor/Warehouse Loader

These set of classes and methods retrieve the data from your RabbitMQ queue, load

them into a work table in the Meta_Warehouse database, and executes your stored procedure

to load the Fact and Dimension Tables

Stored Procedure

Your stored procedure will accept a parameter of @Warehouse_Load_Key INT. You will load

the JSON data from your work table using that load_key, parsing the JSON into a temp table.

You will then do your magic and process the data as required. The entire load should happen

in a transaction so that if it fails, it can be rolled back and the status of event is

recorded in the Warehouse_Load table. If the transaction is successful, the Warehouse_Load

table is updated with a success status instead.

See the Fact_Template stored procedure for a template of how your sproc look.

Manual Testing

Your ETL process can be executed using the **Runner** console app. In Program.cs add a task to the:

```
Task.WaitAll()
```

method.

```
new ProcessFactory(new EtlLogger()).CreateFor("Monitoring.ETL.Domain", "MyDataModel",  
"RabbitMQTemplateModel").RunAsync(null)
```

ThresholdExtractionDelayFactory.cs (Monitoring.ETL.Domain\ThresholdExtractionDelayFactory.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
using System;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain
```

```
{
```

```
    /// <summary>
```

```
    /// Generic Extraction Delay Factory that delays the specified timespan if the total result count is below the given threshold.
```

```
    /// </summary>
```

```
    /// <typeparam name="T">The extract model type.</typeparam>
```

```
    public abstract class ThresholdExtractionDelayFactory<T> : IExtractionDelayFactory<T>
```

```
    {
```

```
        /// <summary>
```

```
        /// The threshold that causes a delay to take place. If the total number of
```

```
        /// results is beyond this value, there is no delay.
```

```
        /// </summary>
```

```
        protected abstract int MaxThresholdForDelay { get; }
```

```
        /// <summary>
```

```
        /// The length of time to delay when the results are under the threshold for delay.
```

```
        /// </summary>
```

```
        protected abstract TimeSpan DelayTimeSpan { get; }
```

```
        public async Task Create(ResultSet<T> results, CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
        {
```

```
            if (results == null || results.Results.Count + results.Exceptions.Count < MaxThresholdForDelay)
```

```
            {
```

```

        logger.Information($"Sleeping for {DelayTimeSpan.TotalSeconds} seconds.");

        await Task.Delay(DelayTimeSpan, cancellationToken);
    }

    else

    {

        logger.Information($"The number of results ({results.Results.Count + results.Exceptions.Count}) was beyond
the max threshold of {MaxThresholdForDelay}. Skipping delay.");

    }

}

}

}

}

```

TimeOfDayExtractionDelayFactory.cs (Monitoring.ETL.Domain\TimeOfDayExtractionDelayFactory.cs):

```

using System;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain
{
    /// <summary>

    /// Generic Extraction Delay Factory that causes the process to run once a day at the specified time.

    /// </summary>

    /// <typeparam name="T">The extract model type.</typeparam>

    public abstract class TimeOfDayExtractionDelayFactory<T> : IExtractionDelayFactory<T>

```

```

{

    /// <summary>

    /// The time of day that the delay should end.

    /// </summary>

    protected abstract TimeSpan TimeOfDay { get; }


    public async Task Create(ResultSet<T> results, CancellationToken cancellationToken, IEtlProcessLogger logger)

    {

        DateTime nextRunTime = DateTime.Today.AddDays(1).Add(TimeOfDay);

        logger.Information($"Sleeping until {nextRunTime}");

        await Task.Delay(nextRunTime.Subtract(DateTime.Now), cancellationToken);

    }

}

}

```

PlexusUser.cs (Monitoring.ETL.Domain\User\PlexusUser.cs):

```

using ETL.Process;


namespace Monitoring.ETL.Domain.User

{

    public class PlexusUser : IExtractModel, SqlJson.IJsonExtractModel

    {

        public string JsonMessage { get; set; }

    }

}

```

PlexusUserDelayFactory.cs (Monitoring.ETL.Domain\User\PlexusUserDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.User
```

```
{
```

```
    public class PlexusUserDelayFactory : TimeOfDayExtractionDelayFactory<PlexusUser>
```

```
    {
```

```
        protected override TimeSpan TimeOfDay => new TimeSpan(3, 30, 00);
```

```
    }
```

```
}
```

PlexusUserExtractor.cs (Monitoring.ETL.Domain\User\PlexusUserExtractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.Warehouse;
```

```
using Monitoring.ETL.Process;
```

```
using Newtonsoft.Json.Linq;
```

```
using Newtonsoft.Json;
```


namespace Monitoring.ETL.Domain.User

```
{  
  
    public class PlexusUserExtractor : IExtractor<PlexusUser>  
  
    {  
  
        private readonly SqlJson.Repository<PlexusUser> _repository;  
  
  
        public PlexusUserExtractor()  
  
        {  
  
            _repository = new SqlJson.Repository<PlexusUser>(new List<string> { EtlSettings.ServerId },  
  
                "Plexus_Control", "");  
  
        }  
  
  
  
        public async Task<ResultSet<PlexusUser>> ExtractAllSinceLastExtractAsync(CancellationTok  
cancellationToken,  
  
            IEtlProcessLogger logger)  
  
        {  
  
            var currentOffset = 0;  
  
            var takeRecords = EtlSettings.BatchSize(5000);  
  
  
            var t2 = @"SELECT DISTINCT PCN  
  
FROM Dim_Customer c  
  
INNER JOIN Dim_Service s ON c.Shard = s.Shard  
  
WHERE s.effective = 1  
  
AND environment = 'Production'  
  
AND service_role = 'Business'  
  
AND s.Data_Center_Abbreviation = '{0}'
```

FOR JSON PATH";

```
var query = string.Format(t2, EtlSettings.UserRegion);
```

```
var warehouseRepo = new WarehouseRepository<PlexusUserRegion>(query);
```

```
logger.Information($"Get PCNs for region {EtlSettings.UserRegion}");
```

```
//this now contains a list of PCNs for this region
```

```
var regionInfo = warehouseRepo.ExecuteWarehouseQuery().Result;
```

```
if (regionInfo == null)
```

```
{
```

```
    logger.Information("No PCNs found in the region");
```

```
    return null;
```

```
}
```

```
logger.Information($"{regionInfo.Count} PCNs found");
```

```
var plexusUserRegions = regionInfo.Select(f => f.PCN).ToList();
```

```
var template = @"SELECT
```

```
PU.Plexus_User_No AS PUN,
```

```
PU.[User_ID],
```

```
PU.Add_Date,
```

```
PU.Active,
```

```
PU.Plexus_Customer_No AS PCN,
```

```
PU.Email
```

```
FROM dbo.Plexus_User AS PU
```

```
ORDER BY Plexus_User_No
```

```
OFFSET {0} ROWS FETCH NEXT {1} ROWS ONLY FOR JSON PATH";
```

```
var resultset = new ResultSet<PlexusUser>();
```

```
var loadMore = true;
```

```
while (loadMore)
```

```
{
```

```
    query = string.Format(template, currentOffset, takeRecords);
```

```
    _repository.SetQueryTemplate(query);
```

```
var result = await _repository.ExtractAllSinceLastExtractAsync(cancellationToken, logger);
```

```
if (result.Results.Count == 0)
```

```
{
```

```
    loadMore = false;
```

```
}
```

```
else
```

```
{
```

```
    currentOffset += takeRecords;
```

```
}
```

```
foreach (var r in result.Results)
```

```
{
```

```
    //only add users from the PCNs in the current region
```

```

        var row = JsonConvert.DeserializeObject(r.JsonMessage) as JObject;

        if (row != null)

        {

            var pcn = Convert.ToInt32(row["PCN"]);

            if (plexusUserRegions.Contains(pcn))

                resultset.Results.Add(r);

        }

    }

    foreach (var e in result.Exceptions)

    {

        resultset.Exceptions.Add(e);

    }

}

return resultset;

}

```

```

public Task<IEnumerable<PlexusUser>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
Cancellation token cancellationToken, IEtlProcessLogger logger)

{

    throw new NotImplementedException();

}

}

}

```

PlexusUserRegion.cs (Monitoring.ETL.Domain\User\PlexusUserRegion.cs):

```
using Monitoring.ETL.Domain.Warehouse;
```

```
namespace Monitoring.ETL.Domain.User
```

```
{
```

```
    internal class PlexusUserRegion : IWarehouseLoadModel
```

```
    {
```

```
        public int PCN { get; set; }
```

```
        /// <summary>
```

```
        /// Not used
```

```
        /// </summary>
```

```
        public int Warehouse_Load_Key { get; set; }
```

```
    }
```

```
}
```

PlexusUserTransformer.cs (Monitoring.ETL.Domain\User\PlexusUserTransformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.User
```

```
{  
  
    public class PlexusUserTransformer : ITransformer<PlexusUser, RabbitMQ.Model.User>  
  
    {  
  
        public async Task<IEnumerable<RabbitMQ.Model.User>> TransformAsync(IEnumerable<PlexusUser> extracted,  
CancellationTokentoken cancellationToken, IEtlProcessLogger logger)  
  
        {  
  
            await Task.Yield();  
  
  
            return extracted.Select(pu =>  
  
                new RabbitMQ.Model.User  
  
                {  
  
                    JsonMessage = pu.JsonMessage  
  
                });  
  
        }  
  
    }  
  
}
```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\User\RabbitMQExtractor.cs):

```
using System;  
  
using System.Collections.Generic;  
  
using System.Threading;  
  
using System.Threading.Tasks;  
  
  
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.User
```

```
{  
  
    public sealed class RabbitMQExtractor : IExtractor<RabbitMQ.Model.User>
```

```
{  
  
    private RabbitMQ.User.Consumer _consumer;
```

```
  
    public async Task<ResultSet<RabbitMQ.Model.User>> ExtractAllSinceLastExtractAsync(CancellationTok  
cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    _consumer = _consumer ?? new RabbitMQ.User.Consumer(logger);  
  
    return await _consumer.ExtractAsync();  
  
}
```

```
  
    public Task<IEnumerable<RabbitMQ.Model.User>> ExtractBetweenAsync(DateTime startDate, DateTime  
endDate, CancellationTok  
cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    throw new NotImplementedException();  
  
}
```

```
}
```

```
}
```

RabbitMQLoader.cs (Monitoring.ETL.Domain\User\RabbitMQLoader.cs):

```
namespace Monitoring.ETL.Domain.User
```

```
{
```

```

public class RabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.User>

{

    public RabbitMQLoader()

        : base(new RabbitMQ.User.Producer())

    {

    }

}

}

```

Repository.cs (Monitoring.ETL.Domain\User\Repository.cs):

```

using Monitoring.ETL.Domain.Warehouse;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Threading;

using System.Threading.Tasks;

using Monitoring.ETL.Process;

```

```

namespace Monitoring.ETL.Domain.User

```

```

{

    internal class Repository

    {

        private readonly ConsulConnectionFactory _connectionFactory;

        private readonly string _consulId;
    }
}

```



```
public Repository(string consuld)
```

```
{
```

```
    _connectionFactory = new ConsulConnectionFactory();
```

```
    _consuld = consuld;
```

```
}
```

```
public async Task<List<PlexusUser>> GetAllAfterDate(CancellationToken cancellationToken, IEtlProcessLogger  
logger)
```

```
{
```

```
    var users = new List<PlexusUser>();
```

```
    using (var connection = _connectionFactory.Create(_consuld, "Plexus_Control"))
```

```
    using (var command = connection.CreateCommand())
```

```
{
```

```
    command.CommandType = System.Data.CommandType.Text;
```

```
    command.CommandText = @"
```

```
USE Plexus_Control;
```

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

```
SELECT
```

```
(
```

```
    SELECT
```

```
        PU.Plexus_User_No AS PUN,
```

```
        PU.[User_ID],
```

```
        PU.Add_Date,
```

```
        PU.Active,
```

```
    PU.Plexus_Customer_No AS PCN,  
  
    PU.Email  
  
FOR JSON PATH,  
  
WITHOUT_ARRAY_WRAPPER  
  
) AS JSON_Message  
  
FROM dbo.Plexus_User AS PU  
  
FOR JSON PATH;"
```

```
try  
  
{  
  
    await connection.OpenAsync(cancellationToken);  
  
    var reader = await command.ExecuteReaderAsync(cancellationToken);  
  
  
    if (reader.HasRows)  
  
    {  
  
        while (await reader.ReadAsync(cancellationToken))  
  
        {  
  
            users.Add(new PlexusUser  
  
            {  
  
                JsonMessage = await Get<string>(reader, 0)  
  
            });  
  
        }  
  
    }  
  
  
    connection.Close();  
  
}
```

```

        catch (Exception e)

        {

            logger.Debug(e.Message);

            throw;

        }

    }

    return users;

}

private async Task<T> Get<T>(SqlDataReader reader, int ordinal)

{

    var value = reader.GetValue(ordinal);

    try

    {

        return (await reader.IsDBNullAsync(ordinal)) ? default(T) : (T)reader.GetValue(ordinal);

    }

    catch (Exception ex)

    {

        return default(T);

    }

}

}

}

```

UserDelayFactory.cs (Monitoring.ETL.Domain\User\UserDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.User
```

```
{
```

```
    public class UserDelayFactory : ThresholdExtractionDelayFactory<RabbitMQ.Model.User>
```

```
    {
```

```
        protected override int MaxThresholdForDelay => 20000;
```

```
        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);
```

```
    }
```

```
}
```

UserTransformer.cs (Monitoring.ETL.Domain\User\UserTransformer.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.User
```

```
{
```

```

public class UserTransformer : ITransformer<RabbitMQ.Model.User, Warehouse.Model.User_w>

{

    public async Task<IEnumerable<Warehouse.Model.User_w>>

TransformAsync(IEnumerable<RabbitMQ.Model.User> extracted, CancellationToken cancellationToken,

IEtlProcessLogger logger)

    {

        await Task.Yield();

        return extracted.Select(pc =>

            new Warehouse.Model.User_w

            {

                JSON_Message = pc.JsonMessage

            });

    }

}

```

WarehouseLoader.cs (Monitoring.ETL.Domain\User\WarehouseLoader.cs):

```

using Monitoring.ETL.Domain.Warehouse;

namespace Monitoring.ETL.Domain.User

{

    public class WarehouseLoader : Loader<Warehouse.Model.User_w>

    {

        public WarehouseLoader() : base("Dim_User_Add")

        {

```

```
    }  
  
    }  
  
}
```

ITaggable.cs (Monitoring.ETL.Domain\Warehouse\ITaggable.cs):

```
using System.Collections.Generic;
```

```
namespace Monitoring.ETL.Domain.Warehouse  
{  
  
    public interface ITaggable : IWarehouseLoadModel  
  
    {  
  
        IEnumerable<Model.Tag_w> Tags { get; set; }  
  
    }  
  
}
```

IWarehouseLoadModel.cs (Monitoring.ETL.Domain\Warehouse\IWarehouseLoadModel.cs):

```
using ETL.Process;
```

```
namespace Monitoring.ETL.Domain.Warehouse  
{  
  
    /// <summary>  
  
    ///  
  
    /// </summary>  
  
    public interface IWarehouseLoadModel : ILoadModel  
  
    {  
  
        /// <summary>
```

```

    /// The load key

    /// </summary>

    int Warehouse_Load_Key { get; set; }

}

}

```

IWarehouseServerProvider.cs (Monitoring.ETL.Domain\Warehouse\IWarehouseServerProvider.cs):

```

namespace Monitoring.ETL.Domain.Warehouse

{

    /// <summary>

    /// Interface for getting the warehouse server.

    /// </summary>

    internal interface IWarehouseServerProvider

    {

        /// <summary>

        /// Gets the name of the warehouse server.

        /// </summary>

        /// <returns></returns>

        string GetWarehouseServerName();

    }

}

```

Loader.cs (Monitoring.ETL.Domain\Warehouse\Loader.cs):

```

using ETL.Process;

using Monitoring.ETL.Process;

```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Data.SqlClient;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.Warehouse
```

```
{
```

```
    public abstract class Loader<T> : ILoader<T> where T : IWarehouseLoadModel
```

```
    {
```

```
        private WarehouseRepository<T> _repository;
```

```
        /// <summary>
```

```
        /// Blank initiation of the warehouse loader. If you call this, you must then use the CreateRepository function to
```

```
        /// initialize your repository.
```

```
        /// </summary>
```

```
        internal Loader()
```

```
        {
```

```
        }
```

```
        /// <summary>
```

```
        /// Initialize a repository with the stored procedure name. The warehouse load key will be passed to the sproc  
        automatically.
```

```
        /// </summary>
```

```
        /// <param name="procedure">The stored procedure name</param>
```


/// <param name="executeProcedure">If true, execute the stored procedure, otherwise skip over it. Helpful when testing or

/// wanting to simply load the warehouse table</param>

internal Loader(string procedure, bool executeProcedure = true)

```
{  
    _repository = new WarehouseRepository<T>(procedure, executeProcedure);  
}
```

/// <summary>

/// Initialize a repository with the stored procedure name and sql parameters. The warehouse load key will be passed to the sproc automatically

/// </summary>

/// <param name="procedure">The stored procedure name</param>

/// <param name="sqlParameters">Additional SQL parameters to the stored procedure.</param>

/// <param name="executeProcedure">If true, execute the stored procedure, otherwise skip over it. Helpful when testing or

/// wanting to simply load the warehouse table</param>

internal Loader(string procedure, SqlParameter[] sqlParameters, bool executeProcedure = true)

```
{  
    _repository = new WarehouseRepository<T>(procedure, sqlParameters, executeProcedure);  
}
```

/// <summary>

/// Load the repository

/// </summary>

/// <param name="transformed"></param>

```

/// <param name="cancellationToken"></param>

/// <param name="logger"></param>

/// <returns></returns>

public async Task<bool> LoadAsync(IEnumerable<T> transformed, CancellationToken cancellationToken,
IEtlProcessLogger logger)

{
    if (_repository == null)
    {
        throw new ArgumentNullException(nameof(_repository),
            "Repository must be initialized either by using the Loader constructor, or by calling the CreateRepository
method");
    }

    return await _repository.LoadAsync(transformed, cancellationToken, logger);
}

/// <summary>

/// Create a new Warehouse repository.

/// </summary>

/// <param name="procedure">The stored procedure name which loads the warehouse table. The warehouse load
key will be passed to the sproc automatically</param>

/// <param name="sqlParameters">Additional SQLParameters to pass to the sproc</param>

/// <param name="executeProcedure">If true, execute the stored procedure, otherwise skip over it. Helpful when
testing or
wanting to simply load the warehouse table</param>

internal void CreateRepository(string procedure, SqlParameter[] sqlParameters, bool executeProcedure = true)

```

```

{

    _repository = new WarehouseRepository<T>(procedure, sqlParameters, executeProcedure);

}

}

}

```

Meta_Warehouse.WorkTables.cs (Monitoring.ETL.Domain\Warehouse\Model\Meta_Warehouse.WorkTables.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
namespace Monitoring.ETL.Domain.Warehouse.Model
```

```

{

    /// <summary>

    /// DO NOT CHANGE THE ORDER OF THESE PROPERTIES. Since the loader uses SqlBulkCopy, the order

    /// determines which columns the data goes into

    /// </summary>

    public partial class Application_Event_w : IWarehouseLoadModel, ITaggable

    {

        public int Warehouse_Load_Key { get; set; }

        public int Load_Event_Number { get; set; }

        public int? Error_Key { get; set; }

        public int? PUN { get; set; }

        public int? PCN { get; set; }

        public DateTime Event_Date { get; set; }

        public string Message { get; set; }

        public string Event_Type { get; set; }
    }
}

```

```
public string Event_Severity { get; set; }

public string Data_Center { get; set; }

public string Environment { get; set; }

public string Service_Application_Name { get; set; }

public string Filename { get; set; }

public string Path_Info { get; set; }

public string HTTP_Referer { get; set; }

public string Referer_Path { get; set; }

public string Web_Server { get; set; }

public string SQL_Server { get; set; }

public string HTTP_User_Agent { get; set; }

public string User_Agent_Browser_Name { get; set; }

public int? User_Agent_Browser_Major_Version { get; set; }

public string User_Agent_OS_Name { get; set; }

public string User_Agent_Device { get; set; }

public string Request_Method { get; set; }

public string Exception_Error_Message { get; set; }

public string Stack_Trace { get; set; }

public string Source_Context { get; set; }

public int? Exception_Data_Source_Key { get; set; }

public string Exception_Data_Source_Name { get; set; }

public string Exception_Column_Name { get; set; }

public string Exception_SQL_Command { get; set; }

public string Exception_Location { get; set; }

public string Element_List { get; set; }

public string Session_Info { get; set; }
```

```

    public string Session_Id { get; set; }

    public string Setting_Group { get; set; }

    public string Setting_Name { get; set; }

    public string Missing_Image_Path { get; set; }

    public string Node_Id { get; set; }

    public int? Thread_Id { get; set; }

    public long? Previous_Change_Version { get; set; }

    public IEnumerable<Tag_w> Tags { get; set; }
}

public partial class Deployment_w : IWarehouseLoadModel
{
    public int Warehouse_Load_Key { get; set; }

    public int? Deployment_Key { get; set; }

    public int? Copy_Of_Deployment_Key { get; set; }

    public int? First_Deployment_Key { get; set; }

    public DateTime Deployment_Date { get; set; }

    public string Deployment_Cart_Name { get; set; }

    public string Repository_Name { get; set; }

    public string Branch_Name { get; set; }

    public string Project_Name { get; set; }

    public string Database_Name { get; set; }

    public string Build_Name { get; set; }

    public string Environment { get; set; }

    public int Owner { get; set; }

    public int Deployed_By { get; set; }
}

```

```
public bool Rollback_Deployment { get; set; }

public bool Duplicate_Deployment { get; set; }

public bool Multi_Line_Package_Deployment { get; set; }

public string Deployment_Tool { get; set; }

public string Deployed_By_User_Id { get; set; }

public string Owner_User_Id { get; set; }

public string Service_Tickets { get; set; }

public int Load_Event_Number { get; set; }

}

public partial class Error_w : IWarehouseLoadModel, ITaggable
{

    public int Warehouse_Load_Key { get; set; }

    public string Elastic_Id { get; set; }

    public int? Error_Key { get; set; }

    public int? PUN { get; set; }

    public int? PCN { get; set; }

    public string Error_Type { get; set; }

    public string Error_Detail { get; set; }

    public string Filename { get; set; }

    public DateTime Error_Date { get; set; }

    public string Cluster_Server { get; set; }

    public string HTTP_Referer { get; set; }

    public string HTTP_User_Agent { get; set; }

    public string Path_Info { get; set; }

    public string Request_Method { get; set; }
```

```
public string Script_Name { get; set; }

public string Lock_Chain { get; set; }

public string SQL_Server { get; set; }

public string Path_Translated { get; set; }

public string Server_Name { get; set; }

public string Element_List { get; set; }

public string Session_Info { get; set; }

public string Setting_Group { get; set; }

public string Setting_Name { get; set; }

public string Session_Key { get; set; }

public int? Exception_Data_Source_Key { get; set; }

public string Exception_Data_Source_Name { get; set; }

public string Exception_Column_Name { get; set; }

public string Exception_SQL_Command { get; set; }

public string Exception_Error_Message { get; set; }

public string User_Agent_Browser_Name { get; set; }

public int? User_Agent_Browser_Major_Version { get; set; }

public string User_Agent_OS_Name { get; set; }

public string User_Agent_Device { get; set; }

public string Missing_Image_Path { get; set; }

public string Stack_Trace { get; set; }

public string Exception_Location { get; set; }

public string Application_Name { get; set; }

public string Referer_Path { get; set; }

public string Event_Severity { get; set; }

public string Data_Center { get; set; }
```

```

    public IEnumerable<Tag_w> Tags { get; set; }

}

public partial class Fact_Data_Size_w : IWarehouseLoadModel
{
    public long Fact_Data_Size_Key { get; set; }

    public int Warehouse_Load_Key { get; set; }

    public int Dim_SQL_Server_Key { get; set; }

    public int Dim_Date_Key { get; set; }

    public int Dim_Time_Key { get; set; }

    public string Data_Base_Name { get; set; }

    public string File_DB_Name { get; set; }

    public string File_Type { get; set; }

    public string Table_Name { get; set; }

    public long? Table_Rows { get; set; }

    public decimal? Table_Reserved_KB { get; set; }

    public decimal? Table_Data_KB { get; set; }

    public decimal? Table_Index_KB { get; set; }

    public decimal? Table_Customer_KB { get; set; }

    public decimal? Table_Unused_KB { get; set; }

    public decimal? File_KB { get; set; }

    public decimal? File_Used_KB { get; set; }

    public decimal? File_Unused_KB { get; set; }

    public decimal? Log_KB { get; set; }

    public decimal? Log_Used_KB { get; set; }

    public decimal? Log_Unused_KB { get; set; }
}

```



```

public int? PCN { get; set; }

public bool? Resident_PCN { get; set; }

public long? PCN_Rows { get; set; }

public decimal? PCN_Customer_KB { get; set; }

public decimal? PCN_Total_KB { get; set; }

}

public partial class Login_w : IWarehouseLoadModel
{
    public int Warehouse_Load_Key { get; set; }

    public int PUN { get; set; }

    public DateTime Login_Date { get; set; }

    public string Source_Server { get; set; }

    public string IP_Address { get; set; }

    public long? Login_No { get; set; }

    public string Login_Origin { get; set; }

    public string Source_SQL_Server_Machine_Name { get; set; }

    public string Source_SQL_Server_Instance_Name { get; set; }

    public int? Source_SQL_Server_Port { get; set; }

}

public partial class ODBC_Statement_w : IWarehouseLoadModel
{
    public int Warehouse_Load_Key { get; set; }

    public long ODBC_Session_Key { get; set; }

    public long ODBC_Session_Statement_Key { get; set; }

```

```

public int? PCN { get; set; }

public int? PUN { get; set; }

public DateTime Session_Start_Date { get; set; }

public DateTime Statement_Start_Date { get; set; }

public int? Equipment_Key { get; set; }

public string Client_Address { get; set; }

public string Custom_Property_List { get; set; }

public string Machine_Name { get; set; }

public bool? Success { get; set; }

public int? Column_Count { get; set; }

public int? Row_Count { get; set; }

public int? Byte_Count { get; set; }

public string SQL_Server_Name { get; set; }

public string SQL_Statement { get; set; }

}

public partial class Performance_Analysis_Data_w : IWarehouseLoadModel
{

    public int Warehouse_Load_Key { get; set; }

    public Int16 Event_Source_Connection_Id { get; set; }

    public Int16 Performance_Analysis_Counter_Id { get; set; }

    public int? Start_Sentry_Timestamp { get; set; }

    public int? End_Sentry_Timestamp { get; set; }

    public float? Metric_Value { get; set; }

    public bool? Amendment_To_Totals { get; set; }

    public int? Sample_Count { get; set; }

```

```
public string Instance_Name { get; set; }
```

```
public string Data_Center { get; set; }
```

```
public string Machine_Name { get; set; }
```

```
public string Server_Instance_Name { get; set; }
```

```
public int Data_Center_Key { get; set; }
```

```
}
```

```
public partial class Plexus_Rendering_Datasource_Json_w : IWarehouseLoadModel
```

```
{
```

```
    public int Warehouse_Load_Key { get; set; }
```

```
    public int Load_Event_Number { get; set; }
```

```
    public string Message_Json { get; set; }
```

```
}
```

```
public partial class Tag_w : IWarehouseLoadModel
```

```
{
```

```
    public int Warehouse_Load_Key { get; set; }
```

```
    public string Elastic_Id { get; set; }
```

```
    public string Tag_Name { get; set; }
```

```
    public int? Load_Event_Number { get; set; }
```

```
}
```

```
public partial class Web_Services_Handler_Json_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public int Load_Event_Number { get; set; }  
  
    public string Message_Json { get; set; }  
  
}
```

```
public partial class Web_Services_Instance_Json_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public int Load_Event_Number { get; set; }  
  
    public string Message_Json { get; set; }  
  
}
```

```
public partial class Customer_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public string JSON_Message { get; set; }  
  
}
```

```
public partial class User_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public string JSON_Message { get; set; }  
  
}
```

```
public partial class Jira_Issues_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public string JSON_Message { get; set; }  
  
}
```

```
public partial class Template_Worktable : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public string JSON_Message { get; set; }  
  
}
```

```
public partial class EDI_Usage_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public string JSON_Message { get; set; }  
  
}
```

```
public partial class Release_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public string JSON_Message { get; set; }  
  
    public int Load_Event_Number { get; internal set; }  
  
}
```

```
public partial class Build_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public string JSON_Message { get; set; }  
  
}
```

```
public partial class Web_Service_Transaction_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public string JSON_Message { get; set; }  
  
}
```

```
public partial class Classic_Data_Source_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public string JSON_Message { get; set; }  
  
}
```

```
public partial class Classic_Screen_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public string JSON_Message { get; set; }  
  
}
```

```
public partial class Cloud_Data_Source_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
}
```

```
    public string JSON_Message { get; set; }

}

public partial class Procedure_w : IWarehouseLoadModel

{

    public int Warehouse_Load_Key { get; set; }

    public string JSON_Message { get; set; }

}

public partial class SQL_Execution_w : IWarehouseLoadModel

{

    public int Warehouse_Load_Key { get; set; }

    public string JSON_Message { get; set; }

}

public partial class Module_w : IWarehouseLoadModel

{

    public int Warehouse_Load_Key { get; set; }

    public string JSON_Message { get; set; }

}

public partial class Impact_Analysis_w : IWarehouseLoadModel

{

    public int Warehouse_Load_Key { get; set; }

    public string JSON_Message { get; set; }

}
```

```
public partial class Windows_Server_Failover_Cluster_Node_Role_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public string Cluster_Name { get; set; }  
  
    public string Node_Name { get; set; }  
  
    public string Node_State { get; set; }  
  
    public string Role_Name { get; set; }  
  
    public string Role_State { get; set; }  
  
    public bool? Core_Role { get; set; }  
  
    public DateTime? State_Timestamp { get; set; }  
  
}
```

```
public partial class Rockwell_Mfg_Index_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public string JSON_Message { get; set; }  
  
}
```

```
public partial class Fact_KPI_DataPoints_w : IWarehouseLoadModel
```

```
{  
  
    public int Warehouse_Load_Key { get; set; }  
  
    public string JSON_Message { get; set; }  
  
}  
  
}
```


Meta_Warehouse.WorkTables.tt (Monitoring.ETL.Domain\Warehouse\Model\Meta_Warehouse.WorkTables.tt):

```
<#@ template debug="false" hostspecific="false" language="C#" #>
```

```
<#@ assembly name="Microsoft.CSharp" #>
```

```
<#@ assembly name="System.Core" #>
```

```
<#@ assembly name="System.Data" #>
```

```
<#@ assembly name="System.Configuration" #>
```

```
<#@ import namespace="System.Linq" #>
```

```
<#@ import namespace="System.Text" #>
```

```
<#@ import namespace="System.Dynamic" #>
```

```
<#@ import namespace="System.Collections.Generic" #>
```

```
<#@ import namespace="System.Configuration" #>
```

```
<#@ import namespace="System.Data.SqlClient" #>
```

```
<#@ output extension=".cs" #>
```

```
using System;
```

```
using System.Collections.Generic;
```

```
namespace <#=System.Runtime.Remoting.Messaging.CallContext.LogicalGetData("NamespaceHint").ToString()#>
```

```
{
```

```
<#
```

```
using (var db = new SqlConnection (@"data source=AH_DBPERF_D01\AH_DBPERF_D01;initial
```

```
catalog=Meta_Warehouse;integrated security=True"))
```

```
using (var cmd = db.CreateCommand())
```

```
{
```

```
    db.Open();
```

```
    IEnumerable<dynamic> tables = ReadRows (cmd, "SELECT * FROM sys.tables ORDER BY name").ToList();
```

```

var columns = ReadRows (cmd, "SELECT C.*, T.name AS type FROM sys.columns AS C JOIN sys.types AS T ON
T.user_type_id = C.system_type_id").ToLookup (k => k.object_id);

foreach (var table in tables.Where(t => t.name.EndsWith("_w")))

{

    var taggable = tables.Any(t => t.name.Equals("Fact_" + table.name.Substring(0, table.name.Length - 2) + "_Tag"));

    var tableColumns = ((IEnumerable<dynamic>)columns[table.object_id]).OrderBy(c => c.column_id).ToList();

#>

public partial class <#=table.name#> : IWarehouseLoadModel<#=(taggable ? ", ITaggable" : "") #>

{

<#

    foreach (var column in tableColumns)

    {

        string type;

        switch (column.type)

        {

            case "bigint":

                type = (column.is_nullable ? "long?" : "long");

                break;

            case "smallint":

                type = (column.is_nullable ? "Int16" : "Nullable<Int16>");

                break;

            case "datetime":

            case "datetime2":

                type = "DateTime";

                break;

```

```

    case "char":

    case "nchar":

    case "varchar":

    case "nvarchar":

    case "sysname":

        type = "string";

        break;

    case "bit":

        type = (column.is_nullable ? "bool?" : "bool");

        break;

    default:

        type = column.type + (column.is_nullable ? "?" : "");

        break;

    }

#>

public <#=type#> <#=column.name#> { get; set; }

<# }

if (taggable)

{

#>

public IEnumerable<Tag_w> Tags { get; set; }

<#}#>

}

<#

```

```
}  
  
}  
  
#>  
  

```

```
IEnumerable<dynamic> ReadRows (SqlCommand command, string sql)
```

```
{  
  
    command.CommandText = sql ?? "";  
  
  
    using (var reader = command.ExecuteReader())  
  
    {  
  
        while (reader.Read())  
  
        {  
  
            var dyn = new ExpandoObject ();  
  
            IDictionary<string, object> dic = dyn;  
  
  
            for (var iter = 0; iter < reader.FieldCount; ++iter)  
  
            {  
  
                dic[reader.GetName(iter) ?? ""] = reader.GetValue(iter);  
  
            }  
  
  
            yield return dyn;  
  
        }  
  
    }  
  
}
```

#>

ModelDataRowMapper.cs (Monitoring.ETL.Domain\Warehouse\ModelDataRowMapper.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Data;
```

```
using System.Data.SqlClient;
```

```
using System.Linq;
```

```
using System.Reflection;
```

```
namespace Monitoring.ETL.Domain.Warehouse
```

```
{
```

```
    public class ModelDataRowMapper<TModel>
```

```
    {
```

```
        private readonly IEnumerable<Action<TModel, DataRow>> _columnMappers;
```

```
        private readonly IEnumerable<Action<SqlBulkCopy>> _columnMappings;
```

```
        private readonly DataTable _table;
```

```
        public ModelDataRowMapper()
```

```
        {
```

```
            var type = typeof(TModel);
```

```
            var properties = type.GetProperties().Where(p =>
```

```
                !p.PropertyType.IsArray && !p.PropertyType.IsInterface && !p.PropertyType.IsAbstract
```

```
            ).ToList();
```

```
_columnMappers = properties
```

```
.Select<PropertyInfo, Action<TModel, DataRow>>(property =>  
    (model, row) => row[property.Name] = property.GetValue(model) ?? DBNull.Value);
```

```
_columnMappings = properties
```

```
.Select<PropertyInfo, Action<SqlBulkCopy>>(property =>  
    sbc => sbc.ColumnMappings.Add(property.Name, property.Name));
```

```
_table = new DataTable();
```

```
foreach (var property in properties)
```

```
{
```

```
    _table.Columns
```

```
        .Add(
```

```
            property.Name,
```

```
            Nullable.GetUnderlyingType(property.PropertyType) ?? property.PropertyType);
```

```
}
```

```
}
```

```
public IEnumerable<DataRow> Map(IEnumerable<TModel> models)
```

```
{
```

```
    var rows = models.Select(model =>
```

```
    {
```

```
        var row = _table.NewRow();
```

```
        foreach (var mapper in _columnMappers)
```

```
        {
```

```

        mapper(model, row);

    }

    return row;

});

_table.Clear();

return rows;

}

public void Map(SqlBulkCopy sbc)
{
    foreach (var mapping in _columnMappings)
    {
        mapping(sbc);
    }
}
}
}

```

WarehouseRepository.cs (Monitoring.ETL.Domain\Warehouse\WarehouseRepository.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
using Newtonsoft.Json.Linq;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Configuration;
```

```
using System.Data;
```

```
using System.Data.SqlClient;
```

```
using System.Linq;
```

```
using System.Runtime.CompilerServices;
```

```
using System.Text;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using Newtonsoft.Json;
```

```
using Monitoring.ETL.Domain.Customer;
```

```
namespace Monitoring.ETL.Domain.Warehouse
```

```
{
```

```
    internal class WarehouseRepository<T> : ILoader<T>
```

```
    where T : IWarehouseLoadModel
```

```
    {
```

```
        private const int SqlTimeoutErrorNumber = -2;
```

```
        private readonly string _connectionString;
```

```
        private readonly bool _executeProcedure;
```

```
        private readonly ModelDataRowMapper<T> _mapper;
```

```
        private readonly string _procedure;
```

```
        private readonly string _query;
```



```
private readonly SqlParameter[] _sqlParameters;
```

```
private readonly string _tableName;
```

```
private readonly WarehouseRepository<Model.Tag_w> _tagRepository;
```

```
private IEtlProcessLogger _logger;
```

```
public WarehouseRepository(string query) : this(new WarehouseServerProvider())
```

```
{
```

```
    _query = query;
```

```
}
```

```
/// <summary>
```

```
/// Create a query with a specific warehouse server. This normally is not used, but useful
```

```
/// when developing and trying to specify dev or prod rather than changing your
```

```
/// registry info
```

```
/// </summary>
```

```
/// <param name="query"></param>
```

```
/// <param name="server"></param>
```

```
public WarehouseRepository(string query, string server) : this(new WarehouseServerProvider())
```

```
{
```

```
    _query = query;
```

```
    _connectionString = GetWarehouseConnectionString(server);
```

```
}
```

```
public WarehouseRepository(string procedure = "", bool executeProcedure = true) : this(new
```

```
WarehouseServerProvider())
```

```
{  
  
    _procedure = procedure;  
  
    _executeProcedure = executeProcedure;  
  
}
```

```
public WarehouseRepository(string procedure, SqlParameter[] sqlParameters, bool executeProcedure = true) :
```

```
this(new WarehouseServerProvider())
```

```
{  
  
    _procedure = procedure;  
  
    _sqlParameters = sqlParameters;  
  
    _executeProcedure = executeProcedure;  
  
}
```

```
internal WarehouseRepository(IWarehouseServerProvider warehouseServerProvider)
```

```
{  
  
    var server = warehouseServerProvider.GetWarehouseServerName();  
  
  
    _mapper = new ModelDataRowMapper<T>();  
  
    _tableName = typeof(T).Name;  
  
  
    if (typeof(T).GetInterfaces().Any(i => i == typeof(ITaggable)))  
  
    {  
  
        _tagRepository = new WarehouseRepository<Model.Tag_w>();  
  
    }  
  
  
    _connectionString = GetWarehouseConnectionString(server);
```

```
}
```

```
internal string GetWarehouseConnectionString(string server)
```

```
{
```

```
    return $"data source={server};initial catalog=Meta_Warehouse;integrated security=True";
```

```
}
```

```
public async Task<bool> LoadAsync(IEnumerable<T> models, CancellationToken cancellationToken,
```

```
IEtlProcessLogger logger)
```

```
{
```

```
    _logger = logger;
```

```
    var warehouseLoadKey = await GetWarehouseLoadKey();
```

```
    return await LoadAsync(models, warehouseLoadKey, logger, cancellationToken);
```

```
}
```

```
private async Task<int> ExecuteWarehouseProcedure(string procedureName, int warehouseLoadKey, params
```

```
SqlParameter[] parameters)
```

```
{
```

```
    using (var connection = new SqlConnection(_connectionString))
```

```
{
```

```
    using (var command = connection.CreateCommand())
```

```
{
```

```
        command.CommandTimeout = EtlSettings.SprocCommandTimeout;
```

```
        var keyParam = command.CreateParameter();
```

```
keyParam.ParameterName = "Warehouse_Load_Key";
```

```
keyParam.DbType = System.Data.DbType.Int32;
```

```
keyParam.Direction = warehouseLoadKey > 0 ? System.Data.ParameterDirection.Input :
```

```
System.Data.ParameterDirection.InputOutput;
```

```
keyParam.Value = warehouseLoadKey;
```

```
command.Parameters.Add(keyParam);
```

```
command.CommandType = System.Data.CommandType.StoredProcedure;
```

```
command.CommandText = procedureName;
```

```
if (parameters != null)
```

```
{
```

```
    foreach (var p in parameters)
```

```
    {
```

```
        if (command.Parameters.Contains(p.ParameterName))
```

```
            command.Parameters.Add(p);
```

```
    }
```

```
}
```

```
await connection.OpenAsync();
```

```
await command.ExecuteNonQueryAsync();
```

```
warehouseLoadKey = Convert.ToInt32(keyParam.Value);
```

```
}
```

```
}
```

```
return warehouseLoadKey;
```

```
}
```

```
private async Task<int> GetWarehouseLoadKey()
```

```
{
```

```
    return await ExecuteWarehouseProcedure("Warehouse_Load_Add", 0);
```

```
}
```

```
/// <summary>
```

```
/// Execute a preset query and deserialize into a list of a given object T
```

```
/// </summary>
```

```
/// <returns></returns>
```

```
public async Task<List<T>> ExecuteWarehouseQuery(bool ignoreMissingJsonMember = false)
```

```
{
```

```
    using (var connection = new SqlConnection(_connectionString))
```

```
{
```

```
    await connection.OpenAsync();
```

```
    using (var command = new SqlCommand(_query, connection))
```

```
{
```

```
    command.CommandType = CommandType.Text;
```

```
    var reader = await command.ExecuteReaderAsync();
```

```
    var json = new StringBuilder();
```

```
    while (await reader.ReadAsync())
```

```
{
```

```
        json.Append(reader.GetValue(0));
```

```
}
```

```
    if (json.Length > 0)
    {
        if (ignoreMissingJsonMember)
        {
            var info = JsonConvert.DeserializeObject<List<T>>(json.ToString());

            return info;
        }
        else
        {
            var jsonSerializerSettings = new JsonSerializerSettings
            {
                MissingMemberHandling = MissingMemberHandling.Ignore,
            };

            var info = JsonConvert.DeserializeObject<List<T>>(json.ToString(), jsonSerializerSettings);

            return info;
        }
    }

    return null;
```

```
}
```

```
private async Task<bool> LoadAsync(IEnumerable<T> models, int warehouseLoadKey, IEtlProcessLogger logger,  
Cancellation token cancellationToken)
```

```
{
```

```
    _logger = logger;
```

```
    var rows = _mapper.Map(models).ToArray();
```

```
    foreach (var row in rows)
```

```
    {
```

```
        row["Warehouse_Load_Key"] = warehouseLoadKey;
```

```
    }
```

```
    var tags = (models as IEnumerable<ITaggable>)?.SelectMany(t => t.Tags).ToList();
```

```
    if (tags != null && tags.Any())
```

```
    {
```

```
        await _tagRepository.LoadAsync(tags, warehouseLoadKey, logger, cancellationToken);
```

```
    }
```

```
while (true)
```

```
{
```

```
    try
```

```
    {
```

```
        using (var connection = new SqlConnection(_connectionString))
```

```
        {
```

```
            var loader = new SqlBulkCopy(connection)
```

```
            {
```

```

        DestinationTableName = _tableName,

        BulkCopyTimeout = 0 // Prevents Timeout

    };

    try

    {

        await connection.OpenAsync(cancellationToken);

        await loader.WriteToServerAsync(rows, cancellationToken);

    }

    finally

    {

        connection.Close();

    }

    break;

}

}

catch (Exception ex)

{

    await SetWarehouseLoadError(warehouseLoadKey, ex, "Error while bulk inserting data.");

    _logger.Information(

        $"Exception during bulk load with Warehouse_Load_Key: {warehouseLoadKey} - {ex.Message}

{ex.StackTrace}. Retrying in 5 seconds.");

    await Task.Delay(TimeSpan.FromSeconds(5), cancellationToken);

}

```



```
}
```

```
while (!string.IsNullOrEmpty(_procedure) && _executeProcedure)
```

```
{
```

```
    try
```

```
    {
```

```
        _logger.Information($"Executing Sproc {_procedure} Load Key: {warehouseLoadKey}");
```

```
        await ExecuteWarehouseProcedure(_procedure, warehouseLoadKey, _sqlParameters);
```

```
        break;
```

```
    }
```

```
    catch (Exception ex)
```

```
    {
```

```
        await SetWarehouseLoadError(warehouseLoadKey, ex, $"Error while executing load procedure:
```

```
{_procedure}");
```

```
        _logger.Information($"Exception during sproc execution: {ex.Message} {ex.StackTrace} retrying in 5
```

```
seconds.");
```

```
        await Task.Delay(TimeSpan.FromSeconds(5), cancellationToken);
```

```
    }
```

```
}
```

```
    return true;
```

```
}
```

```
/// <summary>
```

```
/// Sets the error flag on the Warehouse_Load record.
```

```
/// Also adds exception message text to help troubleshooting.
```

```
/// </summary>
```

```
/// <param name="warehouseLoadKey">The warehouse load key to update.</param>
```

```
/// <param name="ex">The exception to process.</param>
```

```
/// <param name="messagePrefix">String to prefix the message text.</param>
```

```
private async Task SetWarehouseLoadError(int warehouseLoadKey, Exception ex, string messagePrefix)
```

```
{
```

```
    var errorNumber = 0;
```

```
    var timeoutDetected = false;
```

```
    var errorMessageBuilder = new StringBuilder();
```

```
    errorMessageBuilder.Append(messagePrefix)
```

```
        .Append(" - ");
```

```
    var currentException = ex;
```

```
    while (currentException != null)
```

```
{
```

```
    if (currentException != ex)
```

```
{
```

```
        errorMessageBuilder.Append(" INNER EXCEPTION: ");
```

```
}
```

```
    errorMessageBuilder.Append(currentException.Message);
```

```
    if (currentException is SqlException sqlEx)
```

```
{
```

```

        timeoutDetected |= sqlEx.Number == SqlTimeoutErrorNumber;

        foreach (SqlError sqlError in sqlEx.Errors)
        {
            errorMessageBuilder.Append(" SQL ERROR: ")

                .Append(sqlError.Message);

            timeoutDetected |= sqlError.Number == SqlTimeoutErrorNumber;
        }
    }

    currentException = currentException.InnerException;
}

if (timeoutDetected)
{
    errorNumber = SqlTimeoutErrorNumber;
}

await ExecuteWarehouseProcedure("Warehouse_load_Error_Update", warehouseLoadKey,
    new SqlParameter
    {
        ParameterName = "Error_Number",
        DbType = System.Data.DbType.Int32,
        Direction = System.Data.ParameterDirection.Input,
        Value = errorNumber
    }
);

```

```

    },

    new SqlParameter

    {
        ParameterName = "Error_Message",

        DbType = System.Data.DbType.String,

        Direction = System.Data.ParameterDirection.Input,

        Value = errorMessageBuilder.ToString()

    });

}

}

}

```

WarehouseServerProvider.cs (Monitoring.ETL.Domain\Warehouse\WarehouseServerProvider.cs):

```
using Plex.Infrastructure.ConfigSystems;
```

```
using System;
```

```
namespace Monitoring.ETL.Domain.Warehouse
```

```

{

    internal sealed class WarehouseServerProvider : IWarehouseServerProvider

    {

        public string GetWarehouseServerName()

        {

            var registry = new PlexRegistrySystem();

            var server = registry.GetString("CloudOps", "Monitoring", "Warehouse", "Server", null);

```

```

        if (string.IsNullOrEmpty(server))
        {
            throw new InvalidOperationException("The Warehouse server was not found in the registry. Path:
HKLM\\Software\\Plex\\CloudOps\\Monitoring\\Warehouse Value: Server");
        }

        return server;
    }
}
}

```

DelayFactory.cs (Monitoring.ETL.Domain\WebServices\Classic\DelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.WebServices.Classic
```

```

{
    public class DelayFactory : ThresholdExtractionDelayFactory<InstanceMessage>
    {
        protected override int MaxThresholdForDelay => 100;

        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(60);
    }
}

```

Extractor.cs (Monitoring.ETL.Domain\WebServices\Classic\Extractor.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.ElasticSearch;
```

```
using Monitoring.ETL.Process;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using System.Configuration;
```

```
namespace Monitoring.ETL.Domain.WebServices.Classic
```

```
{
```

```
    public class Extractor : IExtractor<InstanceMessage>
```

```
    {
```

```
        private readonly LoadStateRepository<InstanceMessage> _loadStateRepository;
```

```
        private readonly Repository _repository;
```

```
        private DateTime _lastDate;
```

```
        public Extractor()
```

```
        {
```

```
            var serverId = EtlSettings.ServerId;
```

```
            _loadStateRepository = new LoadStateRepository<InstanceMessage>(serverId);
```

```
            _repository = new Repository(serverId);
```

```
        }
```

```
        public async Task<ResultSet<InstanceMessage>> ExtractAllSinceLastExtractAsync(CancellationTok
```

```
        cancellationToken, IEtlProcessLogger logger)
```

```
{

    var resultSet = new ResultSet<InstanceMessage>();

    try

    {

        if (_lastDate == default(DateTime))

        {

            _lastDate = _loadStateRepository.GetDate();

        }

        var transactions = await _repository.GetAllAfterDate(_lastDate, cancellationTokens);

        foreach (var transaction in transactions)

        {

            resultSet.Results.Add(transaction);

        }

        DateTime max = _lastDate;

        if (transactions.Any())

        {

            max = transactions.Max(d => d.TransactionDate);

        }

        if (max > _lastDate)

        {

            _lastDate = max;

        }

    }

}
```

```

        await _loadStateRepository.SetDate(_lastDate);

    }

}

catch (Exception ex)

{

    resultSet.Exceptions.Add(ex);

}

return resultSet;

}

```

```

public Task<IEnumerable<InstanceMessage>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,
Cancellation token cancellationToken, IEtlProcessLogger logger)

{

    throw new NotImplementedException();

}

}

}

```

InstanceMessage.cs (Monitoring.ETL.Domain\WebServices\Classic\InstanceMessage.cs):

```

using ETL.Process;

using System;

```

```

namespace Monitoring.ETL.Domain.WebServices.Classic

```

```

{

    public class InstanceMessage : IExtractModel, SqlJson.IJsonExtractModel

```



```

{

    public DateTime TransactionDate { get; set; }

    public string JsonMessage { get; set; }

}

}

```

RabbitMQTransformer.cs (Monitoring.ETL.Domain\WebServices\Classic\RabbitMQTransformer.cs):

```

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Domain.RabbitMQ.Model;

using Monitoring.ETL.Process;

namespace Monitoring.ETL.Domain.WebServices.Classic

{

    public class RabbitMQTransformer : ITransformer<InstanceMessage, WebServiceTransaction>

    {

        public async Task<IEnumerable<WebServiceTransaction>> TransformAsync(IEnumerable<InstanceMessage>

extracted, CancellationToken cancellationToken, IEtlProcessLogger logger)

        {

            await Task.Yield();

            return extracted.Select(im => new WebServiceTransaction

            {

```

```
        JsonMessage = im.JsonMessage

    });

}

}

}
```

Repository.cs (Monitoring.ETL.Domain\WebServices\Classic\Repository.cs):

```
using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Threading;

using System.Threading.Tasks;

namespace Monitoring.ETL.Domain.WebServices.Classic

{

    internal class Repository

    {

        private readonly ConsulConnectionFactory _connectionFactory;

        private readonly string _consulId;

        private const string _database = "Web_Services";

        public Repository(string consulId)

        {

            _connectionFactory = new ConsulConnectionFactory();

            _consulId = consulId;

        }

    }

}
```

```
public async Task<List<InstanceMessage>> GetAllAfterDate(DateTime date, CancellationToken
cancellationToken)

{
    var transactions = new List<InstanceMessage>();

    using (var connection = _connectionFactory.Create(_consulId, _database))
    using (var command = connection.CreateCommand())
    {
        var minDate = new DateTime(2000, 1, 1);

        date = date > minDate ? date : minDate;

        var keyParam = command.CreateParameter();

        keyParam.ParameterName = "@Last_Date";

        keyParam.DbType = System.Data.DbType.DateTime;

        keyParam.Direction = System.Data.ParameterDirection.Input;

        keyParam.Value = date;

        var countParam = command.CreateParameter();

        countParam.ParameterName = "@Count";

        countParam.DbType = System.Data.DbType.Int32;

        countParam.Direction = System.Data.ParameterDirection.Input;

        countParam.Value = 10000;

        command.CommandType = System.Data.CommandType.Text;

        command.CommandText = @"
```

USE Web_Services;

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

DECLARE @End_Date DATETIME;

SELECT

 @End_Date = I.Receive_Date

FROM dbo.Instance AS I

WHERE I.Receive_Date > @Last_Date

 AND I.Receive_Date < DATEADD(MINUTE, -1, GETDATE())

ORDER BY

 I.Receive_Date

OFFSET (@Count - 1) ROWS FETCH NEXT (1) ROWS ONLY;

IF @End_Date IS NULL

BEGIN

 SELECT

 @End_Date = MAX(I.Receive_Date)

FROM dbo.Instance AS I

WHERE I.Receive_Date > @Last_Date

 AND I.Receive_Date < DATEADD(MINUTE, -1, GETDATE());

END;

WITH Results AS

(

 SELECT

```
M.PCN,

I.Instance_Key,

M.Message_Key,

I.Datasource_Key,

I.Cloud_Data_Source_Key,

I.Host_IP_Address,

I.[Server],

M.Receive_Date,

NULL AS Complete_Date,

DATALENGTH(Message_Xml) AS Message_Length,

MT.Message_Type_Key,

MT.Message_Type,

MT.Module_Key,

H.Handler_Key,

H.[Name] AS Handler_Name,

I.Error

FROM dbo.Instance AS I

JOIN dbo.[Message] AS M

    ON M.PCN = I.PCN

    AND M.Instance_Key = I.Instance_Key

JOIN dbo.Message_Type AS MT

    ON MT.Message_Type_Key = M.Message_Type_Key

JOIN dbo.Handler AS H

    ON H.Handler_Key = MT.Handler_Key

WHERE I.Receive_Date > @Last_Date

    AND I.Receive_Date <= @End_Date
```

AND M.Receive_Date > DATEADD(MINUTE, -1, @Last_Date)

AND M.Receive_Date <= DATEADD(HOUR, 1, @End_Date)

UNION ALL

SELECT

I.PCN,

I.Instance_Key,

NULL AS Message_Key,

I.Datasource_Key,

I.Cloud_Data_Source_Key,

I.Host_IP_Address,

I.Server,

I.Receive_Date,

I.Complete_Date,

0 AS Message_Length,

NULL AS Message_Type_Key,

NULL AS Message_Type,

NULL AS Module_Key,

H.Handler_Key,

H.Name AS Handler_Name,

I.Error

FROM dbo.Instance AS I

JOIN dbo.Handler AS H

ON H.Handler_Key = I.Handler_Key

WHERE I.Receive_Date > @Last_Date

AND I.Receive_Date <= @End_Date

)

SELECT

(

SELECT

WS.PCN,

WS.Instance_Key,

WS.Message_Key,

WS.Datasource_Key,

WS.Cloud_Data_Source_Key,

WS.Host_IP_Address,

WS.Server AS Web_Server,

WS.Receive_Date,

WS.Complete_Date,

WS.Message_Length,

WS.Message_Type_Key,

WS.Message_Type,

WS.Module_Key,

WS.Handler_Key,

WS.Handler_Name,

WS.Error,

@@SERVERNAME AS SQL_Server

FOR JSON PATH,

WITHOUT_ARRAY_WRAPPER

) AS JSON_Message,

WS.Receive_Date AS Transaction_Date

FROM Results AS WS

WHERE WS.Receive_Date<=@End_Date;"

command.Parameters.Add(keyParam);

command.Parameters.Add(countParam);

await connection.OpenAsync();

var reader = await command.ExecuteReaderAsync();

if (reader.HasRows)

{

while (await reader.ReadAsync(cancellationToken))

{

transactions.Add(new InstanceMessage

{

JsonMessage = await Get<string>(reader, 0),

TransactionDate = await Get<DateTime>(reader, 1)

});

}

}

connection.Close();

}

return transactions;

}


```

private async Task<T> Get<T>(SqlDataReader reader, int ordinal)
{
    try
    {
        return (await reader.IsDBNullAsync(ordinal)) ? default(T) : (T)reader.GetValue(ordinal);
    }
    catch (Exception ex)
    {
        return default(T);
    }
}
}
}

```

DatasourceExtractionDelayFactory.cs (Monitoring.ETL.Domain\WebServices\DatasourceExtractionDelayFactory.cs):

```

//using System;

//using Monitoring.ETL.Domain.WebServices.Extract;

//namespace Monitoring.ETL.Domain.Login
//{
//    // public sealed class DatasourceExtractionDelayFactory : ThresholdExtractionDelayFactory<Datasource>
//    // {
//    //     protected override int MaxThresholdForDelay => int.MaxValue; //Always delay so that data is only refreshed daily.
//    //     protected override TimeSpan DelayTimeSpan => TimeSpan.FromDays(1);
//    // }

```

```
//}
```

Datasource.cs (Monitoring.ETL.Domain\WebServices\Extract\Datasource.cs):

```
//using System;
```

```
//using ETL.Process;
```

```
//namespace Monitoring.ETL.Domain.WebServices.Extract
```

```
//{
```

```
// public class Datasource : IExtractModel, SqlJson.IJsonExtractModel
```

```
// {
```

```
// public string JsonMessage { get; set; }
```

```
// }
```

```
//}
```

DatasourceExtractor.cs (Monitoring.ETL.Domain\WebServices\Extract\DatasourceExtractor.cs):

```
//using System;
```

```
//using System.Collections.Generic;
```

```
//using System.Threading;
```

```
//using System.Threading.Tasks;
```

```
//using ETL.Process;
```

```
//using Monitoring.ETL.Process;
```

```
//namespace Monitoring.ETL.Domain.WebServices.Extract
```

```
//{
```

```
// public class DatasourceExtractor : IExtractor<Datasource>
```

```
// {
```

```
// private readonly SqlJson.Repository<Datasource> _extractor;

// public DatasourceExtractor()

// {

//     var consullds = new List<string>

//     {

//         "vp"

//     };

//     var databaseName = "Plexus_Rendering";

//     var query =

//         "SELECT '{consullId}' AS ConsullId, * FROM Datasource WITH (NOLOCK) FOR JSON PATH";

//     _extractor =

//         new SqlJson.Repository<Datasource>(consullds, databaseName, query);

// }

// public async Task<ResultSet<Datasource>> ExtractAllSinceLastExtractAsync(

//     CancellationToken cancellationToken, IEtlProcessLogger logger)

// {

//     return await _extractor.ExtractAllSinceLastExtractAsync(cancellationToken, logger);

// }

// public async Task<IEnumerable<Datasource>> ExtractBetweenAsync(DateTime startDate,

//     DateTime endDate,

//     CancellationToken cancellationToken, IEtlProcessLogger logger)

// {

//     return await _extractor.ExtractBetweenAsync(startDate, endDate, cancellationToken, logger);
```

```
// }  
  
// }  
  
//}
```

WebServicesHandler.cs (Monitoring.ETL.Domain\WebServices\Extract\WebServicesHandler.cs):

```
//using ETL.Process;  
  
//namespace Monitoring.ETL.Domain.WebServices.Extract  
  
//{  
  
// public class WebServicesHandler : IExtractModel, SqlJson.IJsonExtractModel  
  
// {  
  
// public string JsonMessage { get; set; }  
  
// }  
  
//}
```

WebServicesHandlerExtractor.cs (Monitoring.ETL.Domain\WebServices\Extract\WebServicesHandlerExtractor.cs):

```
//using System;  
  
//using System.Collections.Generic;  
  
//using System.Threading;  
  
//using System.Threading.Tasks;  
  
//using ETL.Process;  
  
//using Monitoring.ETL.Process;  
  
//namespace Monitoring.ETL.Domain.WebServices.Extract  
  
//{  
  
// public class WebServicesHandlerExtractor : IExtractor<WebServicesHandler>
```

```
// {

//     private readonly SqlJson.Repository<WebServicesHandler> _extractor;


//     public WebServicesHandlerExtractor()

//     {

//         var consullds = new List<string>

//         {

//             "n1_report",

//             "n2_report",

//             "n3_report",

//             "n4_report",

//             "n5_report",

//             "n6_report",

//             "n7_report",

//             "n8_report",

//             "n1p_report",

//             "n2p_report",

//             "n3p_report",

//             "n4p_report",

//             "n5p_report",

//             "n6p_report",

//             "n7p_report",

//             "n8p_report"

//         };

//         var databaseName = "Web_Services";

//         var query =
```

```

//      "SELECT '{consulId}' AS ConsulId, * FROM Handler WITH (NOLOCK) FOR JSON PATH";

//      _extractor =

//      new SqlJson.Repository<WebServicesHandler>(consulIds, databaseName, query);

//  }


//  public async Task<ResultSet<WebServicesHandler>> ExtractAllSinceLastExtractAsync(
//      CancellationToken cancellationToken, IEtlProcessLogger logger)
//  {

//      return await _extractor.ExtractAllSinceLastExtractAsync(cancellationToken, logger);

//  }


//  public async Task<IEnumerable<WebServicesHandler>> ExtractBetweenAsync(DateTime startDate,
//      DateTime endDate,
//      CancellationToken cancellationToken, IEtlProcessLogger logger)
//  {

//      return await _extractor.ExtractBetweenAsync(startDate, endDate, cancellationToken, logger);

//  }

//  }

//}

```

WebServicesInstance.cs (Monitoring.ETL.Domain\WebServices\Extract\WebServicesInstance.cs):

```

//using ETL.Process;


//namespace Monitoring.ETL.Domain.WebServices.Extract
//{

//  public class WebServicesInstance : IExtractModel, SqlJson.IJsonExtractModel

```

```
// {  
  
//  public string JsonMessage { get; set; }  
  
// }  
  
//}
```

WebServicesInstanceExtractor.cs (Monitoring.ETL.Domain\WebServices\Extract\WebServicesInstanceExtractor.cs):

```
//using System;  
  
//using System.Collections.Generic;  
  
//using System.Threading;  
  
//using System.Threading.Tasks;  
  
//using ETL.Process;  
  
//using Monitoring.ETL.Process;  
  
  
//namespace Monitoring.ETL.Domain.WebServices.Extract  
  
//{  
  
//  public class WebServicesInstanceExtractor : IExtractor<WebServicesInstance>  
  
//  {  
  
//    private readonly SqlJson.Repository<WebServicesInstance> _extractor;  
  
  
//    public WebServicesInstanceExtractor()  
  
//    {  
  
//      var consullds = new List<string>  
  
//      {  
  
//        "n1_report",  
  
//        "n2_report",  
  
//        "n3_report",
```

```
//      "n4_report",

//      "n5_report",

//      "n6_report",

//      "n7_report",

//      "n8_report",

//      "n1p_report",

//      "n2p_report",

//      "n3p_report",

//      "n4p_report",

//      "n5p_report",

//      "n6p_report",

//      "n7p_report",

//      "n8p_report"

//  };

//  var databaseName = "Web_Services";

//  var query =

//      @"

//USE Web_Services;

//SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

//DECLARE @Last_Date DATETIME = '2019-07-19'

//DECLARE @Count INT = 100;

//WITH Results AS

//{
```



```
// SELECT TOP(@Count)

//  M.PCN,

//  I.Instance_Key,

//  M.Message_Key,

//  I.Datasource_Key,

//  I.Cloud_Data_Source_Key,

//  I.Host_IP_Address,

//  I.Server,

//  M.Receive_Date,

//  NULL AS Complete_Date,

//  DATALENGTH(Message_Xml) AS Message_Length,

//  MT.Message_Type_Key,

//  MT.Message_Type,

//  MT.Module_Key,

//  H.Handler_Key,

//  H.Name

// FROM dbo.Message AS M

// JOIN dbo.Instance AS I WITH(FORCESEEK, INDEX(PK_Instance_New))

//  ON I.PCN = M.PCN

//  AND I.Instance_Key = M.Instance_Key

// JOIN dbo.Message_Type AS MT

//  ON MT.Message_Type_Key = M.Message_Type_Key

// JOIN dbo.Handler AS H

//  ON H.Handler_Key = MT.Handler_Key

// WHERE M.Receive_Date > @Last_Date

// ORDER BY
```

```
// M.Receive_Date

// UNION ALL

// SELECT TOP(@Count)

// I.PCN,

// I.Instance_Key,

// NULL AS Message_Key,

// I.Datasource_Key,

// I.Cloud_Data_Source_Key,

// I.Host_IP_Address,

// I.Server,

// I.Receive_Date,

// I.Complete_Date,

// 0 AS Message_Length,

// NULL AS Message_Type_Key,

// NULL AS Message_Type,

// NULL AS Module_Key,

// H.Handler_Key,

// H.Name

// FROM dbo.Instance AS I

// JOIN dbo.Handler AS H

// ON H.Handler_Key = I.Handler_Key

// WHERE I.Receive_Date > DATEADD(MINUTE, -10, @Last_Date)

// AND I.Complete_Date > @Last_Date

// ORDER BY
```

```
// I.Complete_Date

// UNION ALL

// SELECT TOP(@Count)

// I.PCN,

// I.Instance_Key,

// NULL AS Message_Key,

// I.Datasource_Key,

// I.Cloud_Data_Source_Key,

// I.Host_IP_Address,

// I.Server,

// I.Receive_Date,

// I.Complete_Date,

// 0 AS Message_Length,

// NULL AS Message_Type_Key,

// NULL AS Message_Type,

// NULL AS Module_Key,

// H.Handler_Key,

// H.Name

// FROM dbo.Instance AS I

// JOIN dbo.Handler AS H

// ON H.Handler_Key = I.Handler_Key

// WHERE I.Receive_Date > @Last_Date

// AND I.Complete_Date IS NULL

// ORDER BY
```

```

// I.Receive_Date

//)

//SELECT

// *

//FROM

//(

// SELECT

// R.*,

// RANK() OVER(ORDER BY ISNULL(R.Complete_Date, R.Receive_Date)) AS Date_Rank

// FROM Results AS R

//) AS WS

//WHERE WS.Date_Rank <= @Count

//ORDER BY

// WS.Instance_Key,

// ISNULL(WS.Complete_Date, WS.Receive_Date)";

// var batchSize = 10000;

// var checkpointColumn = "Instance_Key";

// var loadNamePrefix = "WebServicesInstance";

// _extractor =

// new SqlJson.Repository<WebServicesInstance>(consulIds, databaseName, query, batchSize,

checkpointColumn, loadNamePrefix);

// }


// public async Task<ResultSet<WebServicesInstance>> ExtractAllSinceLastExtractAsync(

// CancellationToken cancellationToken, IEtlProcessLogger logger)

// {

```

```
// return await _extractor.ExtractAllSinceLastExtractAsync(cancellationTokens, logger);

// }

// public async Task<IEnumerable<WebServicesInstance>> ExtractBetweenAsync(DateTime startDate,
//     DateTime endDate,
//     CancellationToken cancellationToken, IEtlProcessLogger logger)
// {
//     return await _extractor.ExtractBetweenAsync(startDate, endDate, cancellationToken, logger);
// }
// }
// }
```

DatasourceLoadModel.cs (Monitoring.ETL.Domain\WebServices\Load\DatasourceLoadModel.cs):

```
//using ETL.Process;

//using Monitoring.ETL.Domain.RabbitMQ;

//namespace Monitoring.ETL.Domain.WebServices.Load
//{
//     public class DatasourceLoadModel : ILoadModel, IJsonLoadModel
//     {
//         public string JsonMessage { get; set; }
//     }
// }
```

WebServicesHandlerLoadModel.cs (Monitoring.ETL.Domain\WebServices\Load\WebServicesHandlerLoadModel.cs):

```
//using ETL.Process;
```

```
//using Monitoring.ETL.Domain.RabbitMQ;
```

```
//namespace Monitoring.ETL.Domain.WebServices.Load
```

```
//{
```

```
// public class WebServicesHandlerLoadModel : ILoadModel, IJsonLoadModel
```

```
// {
```

```
// public string JsonMessage { get; set; }
```

```
// }
```

```
//}
```

WebServicesHandlerRabbitMQLoader.cs

(Monitoring.ETL.Domain\WebServices\Load\WebServicesHandlerRabbitMQLoader.cs):

```
//using System.Collections.Generic;
```

```
//using System.Threading;
```

```
//using System.Threading.Tasks;
```

```
//using ETL.Process;
```

```
//using Monitoring.ETL.Domain.RabbitMQ.WebServices;
```

```
//using Monitoring.ETL.Process;
```

```
//namespace Monitoring.ETL.Domain.WebServices.Load
```

```
//{
```

```
// public class WebServicesHandlerRabbitMQLoader : ILoader<WebServicesHandlerLoadModel>
```

```
// {
```

```
// private readonly WebServicesHandlerProducer _webServicesHandlerProducer = new
```

```
WebServicesHandlerProducer();
```

```

// public async Task<bool> LoadAsync(IEnumerable<WebServicesHandlerLoadModel> transformed,
//     CancellationToken cancellationToken, IEtlProcessLogger logger)
// {
//     await Task.Run(
//         () =>
//         {
//             foreach (var loadModel in transformed)
//             {
//                 _webServicesHandlerProducer.Publish(loadModel);
//             }
//         });
//
//     return true;
// }
// }
// }
// }

```

WebServicesInstanceLoadModel.cs (Monitoring.ETL.Domain\WebServices\Load\WebServicesInstanceLoadModel.cs):

```

//using ETL.Process;

//using Monitoring.ETL.Domain.RabbitMQ;

//namespace Monitoring.ETL.Domain.WebServices.Load
//{
//     public class WebServicesInstanceLoadModel : ILoadModel, IJsonLoadModel
//     {
//         public string JsonMessage { get; set; }
//     }
// }

```

```

// }

//}

WebServicesInstanceRabbitMQLoader.cs

(Monitoring.ETL.Domain\WebServices\Load\WebServicesInstanceRabbitMQLoader.cs):

//using System.Collections.Generic;

//using System.Threading;

//using System.Threading.Tasks;

//using ETL.Process;

//using Monitoring.ETL.Domain.RabbitMQ.WebServices;

//using Monitoring.ETL.Process;


//namespace Monitoring.ETL.Domain.WebServices.Load

//{{

// public class WebServicesInstanceRabbitMQLoader : ILoader<WebServicesInstanceLoadModel>

// {

//     private readonly WebServicesInstanceProducer _webServicesInstanceProducer = new

WebServicesInstanceProducer();


//     public async Task<bool> LoadAsync(IEnumerable<WebServicesInstanceLoadModel> transformed,

//     CancellationToken cancellationToken, IEtlProcessLogger logger)

//     {

//         await Task.Run(

//             () =>

//             {

//                 foreach (var loadModel in transformed)

```



```
//      {

//      _webServicesInstanceProducer.Publish(loadModel);

//      }

//      });


//      return true;

//  }

// }

//}
```

RabbitMQExtractionDelayFactory.cs (Monitoring.ETL.Domain\WebServices\RabbitMQExtractionDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.WebServices
```

```
{

    public class RabbitMQExtractionDelayFactory :

ThresholdExtractionDelayFactory<RabbitMQ.Model.WebServiceTransaction>

    {

        protected override int MaxThresholdForDelay => 1000;


        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);

    }

}
```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\WebServices\RabbitMQExtractor.cs):

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.WebServices
```

```
{
```

```
    public sealed class RabbitMQExtractor : IExtractor<RabbitMQ.Model.WebServiceTransaction>
```

```
    {
```

```
        private RabbitMQ.WebServices.Consumer _consumer;
```

```
        public async Task<ResultSet<RabbitMQ.Model.WebServiceTransaction>>
```

```
ExtractAllSinceLastExtractAsync(CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
    {
```

```
        _consumer = _consumer ?? new RabbitMQ.WebServices.Consumer(logger);
```

```
        return await _consumer.ExtractAsync();
```

```
    }
```

```
        public Task<IEnumerable<RabbitMQ.Model.WebServiceTransaction>> ExtractBetweenAsync(DateTime startDate,
```

```
DateTime endDate, CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
    {
```

```
        throw new NotImplementedException();
```

```
    }
```

```
}
```

```
}
```

RabbitMQLoader.cs (Monitoring.ETL.Domain\WebServices\RabbitMQLoader.cs):

```
namespace Monitoring.ETL.Domain.WebServices
```

```
{  
  
    public class RabbitMQLoader : RabbitMQ.Loader<RabbitMQ.Model.WebServiceTransaction>  
  
    {  
  
        public RabbitMQLoader()  
  
            : base(new RabbitMQ.WebServices.Producer())  
  
        {  
  
        }  
  
    }  
}
```

RabbitMQWarehouseTransformer.cs (Monitoring.ETL.Domain\WebServices\RabbitMQWarehouseTransformer.cs):

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.WebServices
```

```
{  
  
    public class RabbitMQWarehouseTransformer : ITransformer<RabbitMQ.Model.WebServiceTransaction,  
Warehouse.Model.Web_Service_Transaction_w>  
  
    {  
  
        public async Task<IEnumerable<Warehouse.Model.Web_Service_Transaction_w>>
```

TransformAsync(IEnumerable<RabbitMQ.Model.WebServiceTransaction> extracted, CancellationToken
cancellationToken, IEtlProcessLogger logger)

```
{  
    await Task.Yield();  
  
    return extracted.Select(ws => new Warehouse.Model.Web_Service_Transaction_w  
  
        {  
            JSON_Message = ws.JsonMessage  
        }  
    );  
}  
}
```

DatasourceTransformer.cs (Monitoring.ETL.Domain\WebServices\Transform\DatasourceTransformer.cs):

```
//using System.Collections.Generic;  
  
//using System.Linq;  
  
//using System.Threading;  
  
//using System.Threading.Tasks;  
  
//using ETL.Process;  
  
//using Monitoring.ETL.Domain.WebServices.Extract;  
  
//using Monitoring.ETL.Domain.WebServices.Load;  
  
//using Monitoring.ETL.Process;  
  
  
//namespace Monitoring.ETL.Domain.Login.Transform  
  
//{  
  
//    public sealed class DatasourceTransformer : ITransformer<Datasource, DatasourceLoadModel>
```

```

// {

//     public async Task<IEnumerable<DatasourceLoadModel>> TransformAsync(

//         IEnumerable<Datasource> extracted, CancellationToken cancellationToken, IEtlProcessLogger logger)

//     {

//         return await Task.Run(

//             () => extracted.Select(

//                 i => new DatasourceLoadModel

//                 {

//                     JsonMessage = i.JsonMessage

//                 },

//                 cancellationToken);

//     }

// }

// }

//}

```

WebServicesHandlerTransformer.cs

(Monitoring.ETL.Domain\WebServices\Transform\WebServicesHandlerTransformer.cs):

```

//using System.Collections.Generic;

//using System.Linq;

//using System.Threading;

//using System.Threading.Tasks;

//using ETL.Process;

//using Monitoring.ETL.Domain.WebServices.Extract;

//using Monitoring.ETL.Domain.WebServices.Load;

//using Monitoring.ETL.Process;

```

```
//namespace Monitoring.ETL.Domain.Login.Transform
```

```
//{
```

```
// public sealed class WebServicesHandlerTransformer : ITransformer<WebServicesHandler,  
WebServicesHandlerLoadModel>
```

```
// {
```

```
// public async Task<IEnumerable<WebServicesHandlerLoadModel>> TransformAsync(  

```

```
// IEnumerable<WebServicesHandler> extracted, CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
// {
```

```
// return await Task.Run(  

```

```
// () => extracted.Select(  

```

```
// i => new WebServicesHandlerLoadModel
```

```
// {
```

```
// JsonMessage = i.JsonMessage
```

```
// }),
```

```
// cancellationToken);
```

```
// }
```

```
// }
```

```
//}
```

WebServicesInstanceTransformer.cs

(Monitoring.ETL.Domain\WebServices\Transform\WebServicesInstanceTransformer.cs):

```
//using System.Collections.Generic;
```

```
//using System.Linq;
```

```
//using System.Threading;
```

```
//using System.Threading.Tasks;
```

```
//using ETL.Process;
```

```

//using Monitoring.ETL.Domain.WebServices.Extract;

//using Monitoring.ETL.Domain.WebServices.Load;

//using Monitoring.ETL.Process;


//namespace Monitoring.ETL.Domain.Login.Transform

//{{

// public sealed class WebServicesInstanceTransformer : ITransformer<WebServicesInstance,
WebServicesInstanceLoadModel>

// {

//     public async Task<IEnumerable<WebServicesInstanceLoadModel>> TransformAsync(

//         IEnumerable<WebServicesInstance> extracted, CancellationToken cancellationToken, IEtlProcessLogger logger)

//     {

//         return await Task.Run(

//             () => extracted.Select(

//                 i => new WebServicesInstanceLoadModel

//                 {

//                     JsonMessage = i.JsonMessage

//                 }

//             ),

//             cancellationToken);

//     }

// }

//}}

```

WarehouseLoader.cs (Monitoring.ETL.Domain\WebServices\WarehouseLoader.cs):

```

using Monitoring.ETL.Domain.Warehouse;

```

namespace Monitoring.ETL.Domain.WebServices

```
{  
  
    public class WarehouseLoader : Loader<Warehouse.Model.Web_Service_Transaction_w>  
  
    {  
  
        public WarehouseLoader() : base("Fact_Web_Service_Transaction_Add")  
  
        {  
  
        }  
  
    }  
  
}
```

WebServicesHandlerExtractionDelayFactory.cs

(Monitoring.ETL.Domain\WebServices\WebServicesHandlerExtractionDelayFactory.cs):

```
//using System;  
  
//using Monitoring.ETL.Domain.WebServices.Extract;  
  
  
//namespace Monitoring.ETL.Domain.Login  
  
//{  
  
//    public sealed class WebServicesHandlerExtractionDelayFactory :  
//        ThresholdExtractionDelayFactory<WebServicesHandler>  
  
//    {  
  
//        protected override int MaxThresholdForDelay => int.MaxValue; //Always delay so that data is only refreshed daily.  
  
//        protected override TimeSpan DelayTimeSpan => TimeSpan.FromDays(1);  
  
//    }  
  
//}
```

WebServicesInstanceExtractionDelayFactory.cs

(Monitoring.ETL.Domain\WebServices\WebServicesInstanceExtractionDelayFactory.cs):

```
//using System;

//using Monitoring.ETL.Domain.WebServices.Extract;


//namespace Monitoring.ETL.Domain.Login

//{

// public sealed class WebServicesInstanceExtractionDelayFactory :

ThresholdExtractionDelayFactory<WebServicesInstance>

// {

//     protected override int MaxThresholdForDelay => 50;

//     protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);

// }

//}
```

ClusterExtractor.cs (Monitoring.ETL.Domain\WindowsServerFailoverCluster\ClusterExtractor.cs):

```
using System;

using System.Collections.Concurrent;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using ETL.Process;

using Monitoring.ETL.Domain.Warehouse;

using Monitoring.ETL.Process;
```

```
namespace Monitoring.ETL.Domain.WindowsServerFailoverCluster.Extract
```

```
{  
  
    public sealed class ClusterExtractor : IExtractor<ClusterModel>  
  
    {  
  
        private readonly IWarehouseServerProvider _warehouseServerProvider;  
  
        private readonly ClusterInfoService _clusterService;
```

```
        public ClusterExtractor()  
  
            : this(new WarehouseServerProvider())
```

```
    {  
  
    }
```

```
    internal ClusterExtractor(IWarehouseServerProvider warehouseServerProvider)
```

```
    {  
  
        _warehouseServerProvider = warehouseServerProvider;  
  
        _clusterService = new ClusterInfoService();  
  
    }
```

```
        public Task<IEnumerable<ClusterModel>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,  
CancellationToken cancellationToken, IEtlProcessLogger logger)
```

```
    {  
  
        throw new NotImplementedException();  
  
    }
```

```
        public async Task<ResultSet<ClusterModel>> ExtractAllSinceLastExtractAsync(CancellationToken
```

```
cancellationToken, IEtlProcessLogger logger)
```

```
{
```

```
    var resultSet = new ResultSet<ClusterModel>();
```

```
    Dictionary<string, List<string>> clusterNames = null;
```

```
    try
```

```
    {
```

```
        clusterNames = GetClusterNamesToProcess();
```

```
    }
```

```
    catch (Exception e)
```

```
    {
```

```
        logger.Debug("Unable to get the list of clusters to process");
```

```
        resultSet.Exceptions.Add(e);
```

```
    }
```

```
    if (clusterNames != null)
```

```
    {
```

```
        var concurrentResults = new ConcurrentDictionary<string, ClusterModel>();
```

```
        var options = new ParallelOptions
```

```
        {
```

```
            MaxDegreeOfParallelism = 6,
```

```
            CancellationToken = cancellationToken
```

```
        };
```

```

Parallel.ForEach(clusterNames, options, kvp =>

{

    var clusterName = kvp.Key;

    var nodeNames = kvp.Value;


    var infoLog = new StringBuilder();


    if (_clusterService.TryGetClusterInfo(clusterName, nodeNames, infoLog, out var clusterModel))

    {

        concurrentResults.AddOrUpdate(clusterName, clusterModel, (s, m) => clusterModel);

    }


    logger.Debug(infoLog.ToString());

});


if (cancellationToken.IsCancellationRequested)

{

    logger.Information("Cancellation detected.");

    return null;

}


var resultInfo = new StringBuilder($"Data was collected from {concurrentResults.Count} out of

{clusterNames.Count} clusters.")

    .AppendLine();


foreach (var kvp in clusterNames.OrderBy(s => s.Key))

```

```

{

    var clusterName = kvp.Key;

    if (concurrentResults.TryGetValue(clusterName, out var clusterModel))

    {

        resultSet.Results.Add(clusterModel);

        var nodeCount = clusterModel?.Nodes.Count ?? 0;

        var roleCount = clusterModel?.Nodes.Sum(n => n?.Roles.Count) ?? 0;

        resultInfo.AppendLine($"Cluster: {kvp.Key} - {nodeCount} Node(s) - {roleCount} Roles");

    }

    else

    {

        resultInfo.AppendLine($"Cluster: {kvp.Key} - Unable to collect data.");

    }

}

logger.Information(resultInfo.ToString());

}

return resultSet;

}

/// <summary>

/// Gets cluster names from the CMDB. This is queried from the copy of the data in the performance database.

```

```
/// </summary>
```

```
/// <returns>A dictionary keyed on cluster names. A collection of nodes is returned with each cluster.</returns>
```

```
private Dictionary<string, List<string>> GetClusterNamesToProcess()
```

```
{
```

```
    var clusterNames = new Dictionary<string, List<string>>();
```

```
    var warehouseServerName = _warehouseServerProvider.GetWarehouseServerName();
```

```
    var conStringBuilder = new SqlConnectionStringBuilder
```

```
{
```

```
        InitialCatalog = "Performance",
```

```
        DataSource = warehouseServerName,
```

```
        IntegratedSecurity = true
```

```
};
```

```
using (var con = new SqlConnection(conStringBuilder.ConnectionString))
```

```
{
```

```
    con.Open();
```

```
    using (var cmd = con.CreateCommand())
```

```
{
```

```
        cmd.Connection = con;
```

```
        cmd.CommandType = System.Data.CommandType.StoredProcedure;
```

```
        cmd.CommandText = "dbo.CMDB_Windows_Server_Failover_Clusters_Get";
```

```
        using (var reader = cmd.ExecuteReader())
```

```

{

    if (reader.HasRows)

    {

        int windowsServerFailoverClusterNameOrdinal =

reader.GetOrdinal("Windows_Server_Failover_Cluster_Name");

        int clusterNodeNamesOrdinal = reader.GetOrdinal("Cluster_Node_Names");


        while (reader.Read())

        {

            var clusterName = reader.GetString(windowsServerFailoverClusterNameOrdinal);

            var clusterNodeNames = reader.GetString(clusterNodeNamesOrdinal);


            clusterNames.Add(clusterName, clusterNodeNames.Split(',').ToList());

        }

    }

}

return clusterNames;

}

}

}

```

ClusterInfoService.cs (Monitoring.ETL.Domain\WindowsServerFailoverCluster\ClusterInfoService.cs):

```

using System;

```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Management;
```

```
using System.Runtime.Caching;
```

```
using System.Text;
```

```
namespace Monitoring.ETL.Domain.WindowsServerFailoverCluster
```

```
{
```

```
    /// <summary>
```

```
    /// Service class to gather information about a Windows Server Failover Cluster.=
```

```
    /// </summary>
```

```
    public sealed class ClusterInfoService
```

```
    {
```

```
        private readonly FailedServerCollection _failedServerCollection = new FailedServerCollection();
```

```
        /// <summary>
```

```
        /// Gets information about nodes and roles of the specified cluster.
```

```
        /// This queries WMI to get the cluster information.
```

```
        /// </summary>
```

```
        /// <param name="clusterName">The cluster to get inforamtion about. This is also used to verify that the node is part  
of the desired cluster.</param>
```

```
        /// <param name="nodeNames">The list of possible nodes to query.</param>
```

```
        /// <param name="logger">Logger</param>
```

```
        /// <param name="clusterInfo">The model of the cluster that was looked up.</param>
```

```
        /// <returns>True of the cluster information was retrieved. Else, false.</returns>
```

```
        public bool TryGetClusterInfo(string clusterName, List<string> nodeNames, StringBuilder logger, out ClusterModel
```



```
clusterInfo)
```

```
{
```

```
    logger.AppendLine($"Attempting to connect to a node in cluster {clusterName}");
```

```
    if (TryGetManagementScopeForCluster(clusterName, nodeNames, logger, out var scope))
```

```
    {
```

```
        try
```

```
        {
```

```
            var nodes = GetNodes(scope);
```

```
            var roles = GetRoles(scope);
```

```
            var cluster = new ClusterModel
```

```
            {
```

```
                Name = clusterName.ToUpperInvariant(),
```

```
                Nodes = new List<ClusterNodeModel>(nodes.Count)
```

```
            };
```

```
            foreach (var node in nodes)
```

```
            {
```

```
                var nodeName = (string)node["Name"];
```

```
                var nodeRoles = roles.Where(r => string.Equals((string)r["OwnerNode"], nodeName,
```

```
StringComparison.OrdinalIgnoreCase)).ToList();
```

```
                var clusterNode = new ClusterNodeModel
```

```
                {
```

```

        Name = nodeName,

        State = ConvertNodeStateToDescription((UInt32)node["State"])

    };

    cluster.Nodes.Add(clusterNode);

    clusterNode.Roles = new List<ClusterRoleModel>(nodeRoles.Count);

    foreach (var nodeRole in nodeRoles)
    {
        clusterNode.Roles.Add(new ClusterRoleModel
        {
            Name = (string)nodeRole["Name"],

            IsCore = (bool)nodeRole["IsCore"],

            State = ConvertRoleStateToDescription((UInt32)nodeRole["State"])

        });
    }
}

clusterInfo = cluster;

return true;
}

catch (Exception e)
{
    logger.AppendLine($"Error for cluster {clusterName}. - {e.Message}");

    clusterInfo = null;
}

```

```

        return false;
    }
}

else
{
    logger.AppendLine($"No nodes could be connected to for cluster {clusterName}");

    clusterInfo = null;

    return false;
}
}

private string ConvertRoleStateToDescription(UInt32 state)
{
    // https://docs.microsoft.com/en-us/previous-versions/windows/desktop/cluswmi/mscluster-resourcegroup

    switch (state)
    {
        case 0: return "Online";

        case 1: return "Offline";

        case 2: return "Failed";

        case 3: return "Partial Online";

        case 4: return "Pending";

        default: return "Unknown";
    }
}

private string ConvertNodeStateToDescription(UInt32 state)

```

```

{

// https://docs.microsoft.com/en-us/previous-versions/windows/desktop/cluswmi/mscluster-node

switch (state)

{

case 0: return "Up";

case 1: return "Down";

case 2: return "Paused";

case 3: return "Joining";

default: return "Unknown";

}

}

```

```

private bool TryGetManagementScopeForCluster(string clusterName, List<string> nodeNames, StringBuilder logger,
out ManagementScope managementScope)

{

foreach (var nodeName in nodeNames)

{

if (!_failedServerCollection.ContainsServerName(nodeName))

{

logger.AppendLine($" Skipping server {nodeName} due to a prior failure.");

continue;

}

var path = $"\\{nodeName}\\root\\MSCluster";

var scope = new ManagementScope(path);

```

```
scope.Options.Authentication = AuthenticationLevel.PacketPrivacy;
```

```
try
```

```
{
```

```
    scope.Connect();
```

```
    logger.AppendLine($" ManagementScope connection established to {nodeName}");
```

```
try
```

```
{
```

```
    logger.AppendLine($" Verifying cluster name for node {nodeName}");
```

```
    var clusterNameFromNode = GetClusterName(scope);
```

```
    if (string.Equals(clusterName, clusterNameFromNode, StringComparison.OrdinalIgnoreCase))
```

```
    {
```

```
        managementScope = scope;
```

```
        return true;
```

```
    }
```

```
    else
```

```
    {
```

```
        logger.AppendLine($" The node was not part of the specified cluster. Node: {nodeName} - Expected Cluster:
```

```
{clusterName} - Node's Cluster: {clusterNameFromNode}");
```

```
    }
```

```
}
```

```
catch (Exception e)
```

```
{
```

```
logger.AppendLine($" Unable to verify cluster name for node {nodeName} - {e.Message.Trim()}");
```

```
    _failedServerCollection.AddServerName(nodeName);
```

```
}
```

```
}
```

```
catch (Exception e)
```

```
{
```

```
    logger.AppendLine($" ManagementScope connection failed to {nodeName} - {e.Message.Trim()}");
```

```
    _failedServerCollection.AddServerName(nodeName);
```

```
}
```

```
}
```

```
managementScope = null;
```

```
return false;
```

```
}
```

```
private List<ManagementBaseObject> GetRoles(ManagementScope scope)
```

```
{
```

```
    return GetManagementObjects(scope, "SELECT Name, State, Status, IsCore, OwnerNode FROM
```

```
MSCluster_ResourceGroup");
```

```
}
```

```
private List<ManagementBaseObject> GetNodes(ManagementScope scope)
```

```
{
```

```
    return GetManagementObjects(scope, "SELECT Name, State FROM MScCluster_Node");
```

```
}
```

```

private string GetClusterName(ManagementScope scope)

{

    var clusterNames = GetManagementObjects(scope, "SELECT Name FROM MSCluster_Cluster");


    if (clusterNames.Count > 0)

    {

        // Even though this is a collection, the server can only be part of a single cluster.

        return clusterNames[0]["Name"].ToString();

    }


    return null;

}


private List<ManagementBaseObject> GetManagementObjects(ManagementScope scope, string query)

{

    var objectQuery = new ObjectQuery(query);


    var searcher = new ManagementObjectSearcher(scope, objectQuery);


    var managementObjectCollection = searcher.Get();


    var managementObjects = new List<ManagementBaseObject>();


    if (managementObjectCollection?.Count > 0)

    {

```

```

foreach (var obj in managementObjectCollection)

{

    managementObjects.Add(obj);

}

}


return managementObjects;

}


/// <summary>

/// An internal cache of servers that encountered an issue when connecting.

/// This will evict the server after a 15 minutes.

/// </summary>

private class FailedServerCollection

{

    // Leveraging memory cache to save which nodes were failures.

    // This will expire the entries after the retry time specified below.

    private const string CacheKeyFormat = "ClusterInfoService.FailedServerCollection.FailedServer:{0}";

    private readonly TimeSpan _retryWaitTime = new TimeSpan(0, 15, 0);


    private string GetCacheKey(string serverName)

    {

        return string.Format(CacheKeyFormat, serverName.ToLowerInvariant());

    }


    public bool ContainsServerName(string serverName)

```



```

{

    return MemoryCache.Default.Contains(GetCacheKey(serverName));

}


public void AddServerName(string serverName)

{

    if (ContainsServerName(serverName) == false)

    {

        var cacheItemPolicy = new CacheItemPolicy()

        {

            AbsoluteExpiration = DateTimeOffset.Now.Add(_retryWaitTime)

        };

        MemoryCache.Default.Add(GetCacheKey(serverName), DateTime.Now, cacheItemPolicy);

    }

}

}

}

}

```

ClusterModel.cs (Monitoring.ETL.Domain\WindowsServerFailoverCluster\ClusterModel.cs):

```

using ETL.Process;

using System.Collections.Generic;


namespace Monitoring.ETL.Domain.WindowsServerFailoverCluster

{

```

```
public sealed class ClusterModel : IExtractModel

{

    public string Name { get; set; }


    public List<ClusterNodeModel> Nodes { get; set; }

}

}
```

ClusterModelExtractionDelayFactory.cs

(Monitoring.ETL.Domain\WindowsServerFailoverCluster\ClusterModelExtractionDelayFactory.cs):

```
using System;
```

```
namespace Monitoring.ETL.Domain.WindowsServerFailoverCluster
```

```
{

    public sealed class ClusterModelExtractionDelayFactory : PeriodicExtractionDelayFactory<ClusterModel>

    {

        protected override TimeSpan DelayTimeSpan => new TimeSpan(0, 1, 0);

    }

}
```

ClusterModelTransformer.cs (Monitoring.ETL.Domain\WindowsServerFailoverCluster\ClusterModelTransformer.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Process;
```

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.WindowsServerFailoverCluster
```

```
{
```

```
    public sealed class ClusterModelTransformer : ITransformer<ClusterModel, ClusterNodeRoleModel>
```

```
    {
```

```
        private List<ClusterNodeRoleModel> _priorClusterNodeRoles;
```

```
        public async Task<IEnumerable<ClusterNodeRoleModel>> TransformAsync(IEnumerable<ClusterModel> extracted,  
CancellationTokens cancellationTokens, IEtlProcessLogger logger)
```

```
        {
```

```
            var clusterNodeRoles = new List<ClusterNodeRoleModel>();
```

```
            var timestamp = DateTime.Now;
```

```
            foreach (var cluster in extracted)
```

```
            {
```

```
                foreach (var node in cluster.Nodes)
```

```
                {
```

```
                    if (node.Roles.Where(r => r.IsCore == false).Count() == 0)
```

```
                    {
```

```
                        clusterNodeRoles.Add(new ClusterNodeRoleModel
```

```
                        {
```

```
                            ClusterName = cluster.Name,
```

```
                            NodeName = node.Name,
```

```

        NodeState = node.State,

        RoleName = "[Passive]",

        RoleState = "Online",

        CoreRole = false,

        Timestamp = timestamp

    });

}

foreach (var role in node.Roles)

{

    clusterNodeRoles.Add(new ClusterNodeRoleModel

    {

        ClusterName = cluster.Name,

        NodeName = node.Name,

        NodeState = node.State,

        RoleName = role.Name,

        RoleState = role.State,

        CoreRole = role.IsCore,

        Timestamp = timestamp

    });

}

}

}

if (_priorClusterNodeRoles == null)

{

```

```

        _priorClusterNodeRoles = clusterNodeRoles;

        return clusterNodeRoles;
    }

    var deltaClusterNodeRoles = clusterNodeRoles.Where(c => _priorClusterNodeRoles.Contains(c) == false).ToList();

    logger.Information($"Found {deltaClusterNodeRoles.Count} change(s) since last run.");

    _priorClusterNodeRoles = clusterNodeRoles;

    return deltaClusterNodeRoles;
}
}
}
}

```

ClusterNodeModel.cs (Monitoring.ETL.Domain\WindowsServerFailoverCluster\ClusterNodeModel.cs):

```

using System.Collections.Generic;

namespace Monitoring.ETL.Domain.WindowsServerFailoverCluster
{
    public sealed class ClusterNodeModel
    {
        public string Name { get; set; }

        public string State { get; set; }

        public List<ClusterRoleModel> Roles { get; set; }
    }
}

```

```
}
```

ClusterNodeRoleModel.cs (Monitoring.ETL.Domain\WindowsServerFailoverCluster\ClusterNodeRoleModel.cs):

```
using ETL.Process;
```

```
using Monitoring.ETL.Domain.RabbitMQ;
```

```
using System;
```

```
namespace Monitoring.ETL.Domain.WindowsServerFailoverCluster
```

```
{
```

```
    public sealed class ClusterNodeRoleModel : ILoadModel, IRabbitMQExtractModel
```

```
    {
```

```
        public string ClusterName { get; set; }
```

```
        public string NodeName { get; set; }
```

```
        public string NodeState { get; set; }
```

```
        public string RoleName { get; set; }
```

```
        public string RoleState { get; set; }
```

```
        public bool CoreRole { get; set; }
```

```
        public DateTime Timestamp { get; set; }
```

```
        public override int GetHashCode()
```

```
        {
```

```
            return new Tuple<string, string, string, string, string, bool>(ClusterName, NodeName, NodeState, RoleName,
```

RoleState, CoreRole)

```
.GetHashCode();

}

public override bool Equals(object obj)

{

    if (obj is ClusterNodeRoleModel other)

    {

        return string.Equals(ClusterName, other.ClusterName, StringComparison.OrdinalIgnoreCase)

            && string.Equals(NodeName, other.NodeName, StringComparison.OrdinalIgnoreCase)

            && string.Equals(NodeState, other.NodeState, StringComparison.OrdinalIgnoreCase)

            && string.Equals(RoleName, other.RoleName, StringComparison.OrdinalIgnoreCase)

            && string.Equals(RoleState, other.RoleState, StringComparison.OrdinalIgnoreCase)

            && CoreRole == other.CoreRole;

    }

    return base.Equals(obj);

}

}
```

ClusterNodeRoleModelExtractionDelayFactory.cs

(Monitoring.ETL.Domain\WindowsServerFailoverCluster\ClusterNodeRoleModelExtractionDelayFactory.cs):

using System;

namespace Monitoring.ETL.Domain.WindowsServerFailoverCluster

```

{

    public sealed class ClusterNodeRoleModelExtractionDelayFactory :

ThresholdExtractionDelayFactory<ClusterNodeRoleModel>

    {

        protected override int MaxThresholdForDelay => 10;


        protected override TimeSpan DelayTimeSpan => TimeSpan.FromSeconds(10);

    }

}

```

ClusterRoleModel.cs (Monitoring.ETL.Domain\WindowsServerFailoverCluster\ClusterRoleModel.cs):

```

namespace Monitoring.ETL.Domain.WindowsServerFailoverCluster

{

    public class ClusterRoleModel

    {

        public string Name { get; set; }

        public bool IsCore { get; set; }

        public string State { get; set; }

    }

}

```

RabbitMQExtractor.cs (Monitoring.ETL.Domain\WindowsServerFailoverCluster\RabbitMQExtractor.cs):

```

using ETL.Process;

using Monitoring.ETL.Process;

using System;

using System.Collections.Generic;

```



```
using System.Threading;
```

```
using System.Threading.Tasks;
```

```
namespace Monitoring.ETL.Domain.WindowsServerFailoverCluster
```

```
{  
  
    public sealed class RabbitMQExtractor : IExtractor<ClusterNodeRoleModel>
```

```
{  
  
    private RabbitMQ.WindowsServerFailoverCluster.Consumer _consumer;
```

```
  
    public async Task<ResultSet<ClusterNodeRoleModel>> ExtractAllSinceLastExtractAsync(CancellationToken  
cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    _consumer = _consumer ?? new RabbitMQ.WindowsServerFailoverCluster.Consumer(logger);  
  
    return await _consumer.ExtractAsync();  
  
}
```

```
  
    public Task<IEnumerable<ClusterNodeRoleModel>> ExtractBetweenAsync(DateTime startDate, DateTime endDate,  
CancellationTokn cancellationToken, IEtlProcessLogger logger)
```

```
{  
  
    throw new NotImplementedException();  
  
}  
  
}  
  
}
```

RabbitMQLoader.cs (Monitoring.ETL.Domain\WindowsServerFailoverCluster\RabbitMQLoader.cs):

```
namespace Monitoring.ETL.Domain.WindowsServerFailoverCluster
```

```

{

public sealed class RabbitMQLoader : RabbitMQ.Loader<ClusterNodeRoleModel>

{

public RabbitMQLoader()

: base(new RabbitMQ.WindowsServerFailoverCluster.Producer())

{

}

}

}

```

RabbitMQWarehouseTransformer.cs

(Monitoring.ETL.Domain\WindowsServerFailoverCluster\RabbitMQWarehouseTransformer.cs):

```

using ETL.Process;

using Monitoring.ETL.Domain.Warehouse.Model;

using Monitoring.ETL.Process;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;


namespace Monitoring.ETL.Domain.WindowsServerFailoverCluster

{

public sealed class RabbitMQWarehouseTransformer : ITransformer<ClusterNodeRoleModel,

Windows_Server_Failover_Cluster_Node_Role_w>

{

public async Task<IEnumerable<Windows_Server_Failover_Cluster_Node_Role_w>>

```

TransformAsync(IEnumerable<ClusterNodeRoleModel> extracted, CancellationToken cancellationToken,

IEtlProcessLogger logger)

```
{  
  
    return extracted.Select(e => new Windows_Server_Failover_Cluster_Node_Role_w  
  
    {  
  
        Cluster_Name = e.ClusterName,  
  
        Node_Name = e.NodeName,  
  
        Node_State = e.NodeState,  
  
        Role_Name = e.RoleName,  
  
        Role_State = e.RoleState,  
  
        Core_Role = e.CoreRole,  
  
        State_Timestamp = e.Timestamp  
  
    });  
  
}  
  
}  
  
}
```

WarehouseLoader.cs (Monitoring.ETL.Domain\WindowsServerFailoverCluster\WarehouseLoader.cs):

using Monitoring.ETL.Domain.Warehouse;

namespace Monitoring.ETL.Domain.WindowsServerFailoverCluster

```
{  
  
    public sealed class WarehouseLoader : Loader<Warehouse.Model.Windows_Server_Failover_Cluster_Node_Role_w>  
  
    {  
  
        public WarehouseLoader()  
  
            : base("Dim_Windows_Server_Failover_Cluster_Node_Role_ETL")  
  
        }  
    }
```

{

}

}

}