﻿<#@ template debug="false" hostspecific="true" language="C#" #>

<#@ output extension= "/" #>

<#@ include file="MultipleOutputHelper.ttinclude" #>

<#@ assembly name="System.Core" #>

<#@ assembly name="System.Data" #>

<#@ assembly name="Microsoft.SqlServer.Smo" #>

<#@ assembly name="Microsoft.SqlServer.ConnectionInfo" #>

<#@ assembly name="Microsoft.SqlServer.Management.Sdk.Sfc" #>

<#@ import namespace="System" #>

<#@ import namespace="System.Linq" #>

<#@ import namespace="System.Data" #>

<#@ import namespace="System.Text" #>

<#@ import namespace="System.Collections.Generic" #>

<#@ import namespace="System.Data.SqlClient" #>

<#@ import namespace="Microsoft.SqlServer.Management.Common" #>

<#@ import namespace="Microsoft.SqlServer.Management.Smo" #>

<#@ include file="includeTables.ttinclude" #>

<#

// NOTE:

// tablesToInclude is sourced from the includeTables.ttinclude file

List<string> includeTables = new List<string>();

includeTables.AddRange(includeStaticTables);

includeTables.AddRange(includedDynamicTables);

//\*\*\*\*\*\*\*\*\*\*DISABLE LAZY LOADING?\*\*\*\*\*\*\*\*\*\*

bool disableLazyLoading = true;

//\*\*\*\*\*\*\*\*\*\*TEMPORAL BATCH\*\*\*\*\*\*\*\*\*\*\*\*\*

includeTables.Add("TemporalBatch");

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

string[] tablesToInclude = includeTables.ToArray();

Utility.TablesWithUniquePks = \_tablesWithUniquePks.ToList();

string[] columnsToIgnore = new string[] { "StartTime", "EndTime" };

string[] schemasToExclude = new string[] {"Hangfire", "Spatial", "History"};

string[] tablesToExclude = new string[] {"SpatialBoundary"};

Utility.MaxLengthDataTypes = new List<string> {"varchar", "char", "nchar", "ntext", "text", "xml", "decimal"};

Utility.Tables = new List<Table>();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Is this an IdentityDbContext? \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

bool isIdentityContext = false;

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Set up connection and File Info \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

string connectionString = "Server=.;trusted\_connection=true";

string sqlDatabase = "MyDB\_Local";

// The namespace to apply to the generated classes

string classNamespace = "BaseNamespace.Data.Entities";

// The destination folder for the generated classes, relative to this file's location.

string destinationFolder = "";

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

var manager = Manager.Create(Host, GenerationEnvironment);

// Loop over each table and create a class file!

SqlConnection connection = new System.Data.SqlClient.SqlConnection(connectionString);

ServerConnection conn = new ServerConnection(connection);

Server server = new Server(conn);

server.ConnectionContext.Connect();

foreach (Table table in server.Databases[sqlDatabase].Tables)

{

// Skip tables

if(table.Name != "TemporalBatch" && ( tablesToInclude.Length != 0 && !tablesToInclude.Contains(table.Name) || schemasToExclude.Contains(table.Schema)))

{

continue;

}

if(!Utility.Tables.Any(i=> i.Name == table.Name))

{

Utility.Tables.Add(table);

}

}

manager.StartNewFile("dbContext.cs");

#>

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Data.Entity.SqlServer;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data.Common;

using System.Data.Entity.Infrastructure;

using System.Data.Entity.Validation;

<# if (isIdentityContext)

{#>

using Microsoft.AspNet.Identity.EntityFramework;

<#}#>

using ESS.Data.Entities;

namespace ESS.Data

{

public partial class ESSDbContext : <#= isIdentityContext? "IdentityDbContext<User>" : "DbContext" #>

{

public ESSDbContext() : base("ESSEntities")

{

<#=!disableLazyLoading?"//":string.Empty#>Configuration.LazyLoadingEnabled = false;

}

public ESSDbContext(string connectionName) : base(connectionName)

{

<#=!disableLazyLoading?"//":string.Empty#>Configuration.LazyLoadingEnabled = false;

}

<# foreach(Table table in Utility.Tables)

{

#>

public virtual DbSet<<#=table.Name #>> <#=Utility.Pluralize(table.Name) #> { get; set; }

<#

}

#>

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

CustomOnModelCreating(modelBuilder);

#region Base Tables

<# foreach(Table table in Utility.Tables.Where(i=> i.Schema != "Lookup"))

{

if(!Utility.TablesWithUniquePks.Contains(table.Name))

{

#>

modelBuilder

.Entity<<#=table.Name #>>()

.ToTable("<#=table.Name #>", "<#=table.Schema #>")

.HasKey(<#=table.Name.ToLower() #> => <#=table.Name.ToLower() #>.Id)

.Property(<#=table.Name.ToLower() #> => <#=table.Name.ToLower() #>.Id)

.HasColumnName("<#=table.Name #>Id")

<# if(table.Columns[table.Name +"Id"].Identity)

{

#>

.HasDatabaseGeneratedOption(System.ComponentModel.DataAnnotations.Schema.DatabaseGeneratedOption.Identity);

<# } else {

#>

.HasDatabaseGeneratedOption(System.ComponentModel.DataAnnotations.Schema.DatabaseGeneratedOption.None);

<# }

#>

<#

}

else

{

#>

modelBuilder

.Entity<<#=table.Name #>>()

.ToTable("<#=table.Name #>", "<#=table.Schema #>")

.HasKey(<#=table.Name.ToLower() #> => <#=table.Name.ToLower() #>.<#=table.Name #>Id)

.Property(<#=table.Name.ToLower() #> => <#=table.Name.ToLower() #>.<#=table.Name #>Id);

<#

}

}

#>

#endregion Base Tables

#region Associations

<#

foreach(Table table in Utility.Tables)

{

foreach (ForeignKey key in table.ForeignKeys)

{

foreach (ForeignKeyColumn column in key.Columns)

{

#>

//<#=key.ToString() #>

<#

}

}

}

#>

#endregion Associations

OnModelCreatingWithIdentities(modelBuilder);

base.OnModelCreating(modelBuilder);

}

protected virtual void OnModelCreatingWithIdentities(DbModelBuilder modelBuilder)

{

<# foreach(Table table in Utility.Tables.Where(i=> i.Schema == "Lookup"))

{

#>

modelBuilder

.Entity<<#=table.Name #>>()

.ToTable("<#=table.Name #>", "<#=table.Schema #>")

.HasKey(<#=table.Name.ToLower() #> => <#=table.Name.ToLower() #>.Id)

.Property(<#=table.Name.ToLower() #> => <#=table.Name.ToLower() #>.Id)

.HasColumnName("<#=table.Name #>Id")

.HasDatabaseGeneratedOption(System.ComponentModel.DataAnnotations.Schema.DatabaseGeneratedOption.Identity);

<#

}

#>

}

partial void CustomOnModelCreating(DbModelBuilder modelBuilder);

}

}

<#

manager.Process(true);

//Helper Methods

#>

<#+

public static class Utility

{

//public static Dictionary<string,string> ChildCollections;

public static List<string> TablesWithUniquePks = new List<string>();

public static List<Table> Tables = new List<Table>();

public static List<string> MaxLengthDataTypes;

public static List<Column> Columns(Table t)

{

List<Column> columns = new List<Column>();

foreach (Column c in t.Columns)

{

columns.Add(c);

}

return columns;

}

public static string ClassDeclaration(string accessModifier, Table table)

{

List<Column> columns = Columns(table);

Column pk = columns.SingleOrDefault(i => i.InPrimaryKey);

string pkType = GetClrType(pk.DataType.SqlDataType, pk.Nullable);

StringBuilder className = new StringBuilder();

className.Append(accessModifier);

className.Append(" class ");

className.Append(table.Name);

className.Append(" : BaseEntity<");

className.Append(pkType);

className.Append(">, IBaseEntity<");

className.Append(pkType);

className.Append(">");

if (ContainsColumns(columns, new string[] { "IsDeleted" }))

{

className.Append(", ILogicalDelete");

}

if (ContainsColumns(columns, new string[] { "CreatedBy", "CreatedDate", "LastUpdated", "UpdatedBy" }))

{

className.Append(", IAudit");

}

if (ContainsColumns(columns, new string[] { "AddressId", "AddressTypeId" }))

{

className.Append(", IAddressAssociation");

}

if (ContainsColumns(columns, new string[] { "EffectiveDate", "ExpirationDate" }))

{

className.Append(", IHistorical");

}

if (ContainsColumns(columns, new string[] { "Code", "Name", "Description" }))

{

className.Append(", ITypeEntity<");

className.Append(pkType);

className.Append(">");

}

//Applicant : BasePOCO<int>, IBasePOCO<int>, ILogicalDelete, IAudit

return className.ToString();

}

public static bool ContainsColumns(List<Column> columns, string[] columnNames)

{

int columnsContained = 0;

foreach (string column in columnNames)

{

if (columns.Any(i => i.Name == column))

columnsContained++;

}

return columnsContained == columnNames.Length;

}

public static string PropertyDeclaration(Column c)

{

StringBuilder declaration = new StringBuilder();

if(c.InPrimaryKey && !TablesWithUniquePks.Contains((c.Parent as Table).Name))

{

declaration.Append("[Required]");

declaration.Append(Environment.NewLine);

declaration.Append(Indent("public override ",2));

declaration.Append(GetClrType(c.DataType.SqlDataType, c.Nullable));

declaration.Append(" ");

declaration.Append(" Id");

}

else

{

if (!c.Nullable)

{

declaration.Append("[Required]");

declaration.Append(Environment.NewLine);

declaration.Append(Indent("",2));

}

if (MaxLengthDataTypes.Contains(c.DataType.Name))

{

declaration.Append("[MaxLength(");

declaration.Append(c.DataType?.MaximumLength.ToString());

declaration.Append(")]");

declaration.Append(Environment.NewLine);

declaration.Append(Indent("",2));

}

if (c.DataType.Name.ToLower() == "decimal")

{

declaration.Append("[RegularExpression(@\"\\d+(\\.\\d{1,");

declaration.Append(c.DataType.NumericPrecision.ToString());

declaration.Append("})?\", ErrorMessage = \"Decimal Value has a precision of ");

declaration.Append(c.DataType.NumericPrecision.ToString());

declaration.Append("]");

declaration.Append(Environment.NewLine);

declaration.Append(Indent("",2));

}

declaration.Append("public ");

declaration.Append(GetClrType(c.DataType.SqlDataType, c.Nullable));

declaration.Append(" ");

declaration.Append(c.Name);

}

return declaration.ToString();

}

public static string GenerateForeignKey(ForeignKey key, ForeignKeyColumn column)

{

StringBuilder declaration = new StringBuilder();

declaration.Append("[ForeignKey(\"");

declaration.Append(column.Name);

declaration.Append("\")]");

declaration.Append(Environment.NewLine);

declaration.Append(Indent("public virtual ", 2));

declaration.Append(key.ReferencedTable);

declaration.Append(" ");

declaration.Append(column.Name.EndsWith("Id")?column.Name.Remove(column.Name.Length-2,2):key.ReferencedTable);

declaration.Append(" { get; set; } ");

return declaration.ToString();

}

public static string GetClrType(SqlDataType sqlType, bool isNullable)

{

switch (sqlType)

{

case SqlDataType.BigInt:

return isNullable? "long?" : "long";

case SqlDataType.Binary:

case SqlDataType.Image:

case SqlDataType.Timestamp:

case SqlDataType.VarBinary:

case SqlDataType.VarBinaryMax:

return "byte[]";

case SqlDataType.Bit:

return isNullable? "bool?" : "bool";

case SqlDataType.Char:

case SqlDataType.NChar:

case SqlDataType.NText:

case SqlDataType.NVarChar:

case SqlDataType.NVarCharMax:

case SqlDataType.Text:

case SqlDataType.VarChar:

case SqlDataType.Xml:

return "string";

case SqlDataType.DateTime:

case SqlDataType.SmallDateTime:

case SqlDataType.Date:

case SqlDataType.Time:

case SqlDataType.DateTime2:

return isNullable? "DateTime?" : "DateTime";

case SqlDataType.Decimal:

case SqlDataType.Money:

case SqlDataType.SmallMoney:

return isNullable? "decimal?" : "decimal";

case SqlDataType.Float:

return isNullable? "double?" : "double";

case SqlDataType.Int:

return isNullable? "int?" : "int";

case SqlDataType.Real:

return isNullable? "float?" : "float";

case SqlDataType.UniqueIdentifier:

return isNullable? "Guid?" : "Guid";

case SqlDataType.SmallInt:

return isNullable? "short?" : "short";

case SqlDataType.TinyInt:

return isNullable? "byte?" : "byte";

case SqlDataType.DateTimeOffset:

return isNullable? "DateTimeOffset?" : "DateTimeOffset";

default:

throw new ArgumentOutOfRangeException("sqlType");

}

}

public static string Indent(string value)

{

return " " + value;

}

public static string Indent(string value, int numberOfIndents)

{

for (int i = 0; i < numberOfIndents; i++)

{

value = Indent(value);

}

return value;

}

public static string Outdent(string value)

{

if (value.StartsWith(" "))

value.Remove(0, 4);

return value;

}

public static string GetForeignKeyPropertyName(string foreignKeyName)

{

string[] nameParts = foreignKeyName.Split('\_');

if(nameParts.Length == 4)

{

return nameParts[3];

}

else return null;

}

public static string GenerateChildCollection(string childTableName, string fkName)

{

string name = GetForeignKeyPropertyName(fkName) ?? childTableName;

StringBuilder declaration = new StringBuilder();

declaration.Append("public virtual ICollection<");

declaration.Append(childTableName);

declaration.Append("> ");

declaration.Append(Pluralize(name));

declaration.Append(" { get; set; }");

return declaration.ToString();

}

//public static void ClearChildCollection()

//{

// ChildCollections = new Dictionary<string, string>();

//}

public static string Pluralize(string value )

{

string[] isPlural = {"fish", "data"};

if(isPlural.Contains(value.ToLower()))

return value;

if (value.EndsWith("ess"))

return value +"Records";

if (value.EndsWith("ies"))

return value;

if (value.EndsWith("es"))

return value;

if (value.EndsWith("s") || value.EndsWith("tch"))

return value + "es";

if (value.EndsWith("ey"))

return value +"s";

if (value.EndsWith("y"))

return value.Remove(value.Length - 1, 1) + "ies";

else

return value + "s";

}

}

#>