**Программная генерация скриптов для MSSQL**

[habrahabr](http://www.itshop.ru/go.asp?url=-3D-41-52-17-22-18-3E-2B-DD-B6-8A-20-3F-9B-80-9D-D3-DC-DF-08-07-72-C3-F3-28-50-0F-51-EE-7C-33-A8)

RedQuark

В силу частых и неупорядоченных изменений базы данных, большим числом пользователей, часто возникает вопросы о истории изменений. Речь не идет о тотально логирование всех изменений, которые происходят с базой в течение дня. Интерес представляют собой снимки структуры БД каждый день после окончания рабочего дня. С помощью **SQL Server Management Studio** можно сгенерировать скрипты, но поштучно или все сразу. Полную свободу действий можно получить использовав набор библиотек от [**SQL Server Management Studio**](http://www.itshop.ru/Microsoft-Servernye-programmy/SQL-Server/l2t1f0c433sc68) в вашем .NET приложение. Описание программы генерации скриптов: таблиц, представлений, процедур далее.

**Присоединение библиотек**

Потребуются классы из пространств имен:

using Microsoft.SqlServer.Management.Smo;

using Microsoft.SqlServer.Management.Common;

using Microsoft.SqlServer.Management.Sdk.Sfc;

Библиотеки, которые их содержат имеют такие же названия и находятся в папке:  
C:\Program Files\Microsoft SQL Server\90\SDK\Assemblies  
или  
C:\Program Files\Microsoft SQL Server\100\SDK\Assemblies  
в зависимости от версии SQL Server.

**Выполняемый сценарий**

Генератор скриптов создает отдельные папки Tables, Views, Procedures в указанной директории. Создает скрипты создания объектов и сохраняет в отдельных файлах в соответствующих папках. Для таблиц генерация производиться с учетом зависимостей(ключи, индексы и т.д.) и без. Создает общий файл однотипных скриптов в указанной директории.

//Создается экземпляр сервера

Server myServer = new Server(@"myServ");

//Аутентификация Windows

myServer.ConnectionContext.LoginSecure = true;

//Открыть соединение

myServer.ConnectionContext.Connect();

//Директория создается автоматически, с новой папкой на каждый день

string dir = Application.StartupPath +@"\"+ DateTime.Now.Day.ToString() + "\_" + DateTime.Now.Month.ToString() + "\_" + DateTime.Now.Year.ToString();

Directory.CreateDirectory(dir);

//Генерация таблиц, описание ниже

GenerateTableScript(myServer,dir);

//Генерация процедур, описание ниже

GenerateProceduresScript(myServer, dir);

//Генерация представлений, описание ниже

GenerateViewScript(myServer, dir);

//Закрыть соединение

myServer.ConnectionContext.Disconnect();

**Генерация скриптов для таблиц**

private static void GenerateTableScript(Server myServer, string path)

{

Directory.CreateDirectory(path + @"\Tables\");

string text = "";

string textWithDependencies = "";

//Создаем экземпляр класса, который будет генерировать скрипты

Scripter scripter = new Scripter(myServer);

//Создаем экземпляр класса базы данных, "ZZZ" - название базы данных

Database myAdventureWorks = myServer.Databases["ZZZ"];

//Создаем экземпляр класса настроек генерации скриптов

ScriptingOptions scriptOptions = new ScriptingOptions();

//Функциональность свойств у класса настроек генерации легко определяема

//Не создавать скрипт с Drop

scriptOptions.ScriptDrops = false;

//Не включать скрипт с If Not Exists

scriptOptions.IncludeIfNotExists = false;

//Перебираем все таблицы

foreach (Table myTable in myAdventureWorks.Tables)

{

//Получаем sql запрос на основание выбранных параметров

StringCollection tableScripts = myTable.Script(scriptOptions);

//Переменная для объединения строк

string newSql = "";

//Объединяем строки

foreach (string script in tableScripts)

{

newSql = newSql + script;

text = text + script;

}

//Записываем в файл скрипт создания таблицы без зависимостей

File.WriteAllText(path + @"\Tables\" + myTable.Name + ".sql", newSql);

//Определяем новые параметры генерации

scriptOptions.DriAllConstraints = true;

scriptOptions.DriAllKeys = true;

scriptOptions.DriDefaults = true;

scriptOptions.DriForeignKeys = true;

scriptOptions.DriIndexes = true;

scriptOptions.DriNonClustered = true;

scriptOptions.DriPrimaryKey = true;

scriptOptions.DriUniqueKeys = true;

tableScripts = myTable.Script(scriptOptions);

newSql = "";

foreach (string script in tableScripts)

{

newSql = newSql + script;

textWithDependencies = text + script;

}

//Записываем в файл скрипт создания таблицы с зависимостями

File.WriteAllText(path + @"\Tables\" + myTable.Name + "\_WithDependencies.sql", newSql);

}

//Записываем общие объединяющие файлы

File.WriteAllText(path + @"\" + "AllTable\_WithDependencies.sql", text);

File.WriteAllText(path + @"\" + "AllTable.sql", text);

}

**Генерация скриптов для представлений**

private static void GenerateViewScript(Server myServer, string path)

{

Directory.CreateDirectory(path + @"\View\");

string text = "";

Scripter scripter = new Scripter(myServer);

Database myAdventureWorks = myServer.Databases["ZZZ"];

ScriptingOptions scriptOptions = new ScriptingOptions();

scriptOptions.ScriptDrops = false;

scriptOptions.IncludeIfNotExists = false;

foreach (Microsoft.SqlServer.Management.Smo.View myView in myAdventureWorks.Views)

{

StringCollection ProcedureScripts = myView.Script(scriptOptions);

ProcedureScripts = myView.Script();

string newSql = "";

foreach (string script in ProcedureScripts)

{

newSql = newSql + script;

text = text + script;

}

File.WriteAllText(path + @"\Views\" + myView.Name + ".sql", newSql);

}

File.WriteAllText(path + @"\" + "AllView.sql", text);

}

**Генерация скриптов для процедур**

private static void GenerateProceduresScript(Server myServer, string path)

{

Directory.CreateDirectory(path + @"\Procedures\");

string text = "";

Scripter scripter = new Scripter(myServer);

Database myAdventureWorks = myServer.Databases["ZZZ"];

ScriptingOptions scriptOptions = new ScriptingOptions();

scriptOptions.ScriptDrops = false;

scriptOptions.IncludeIfNotExists = false;

foreach (StoredProcedure myProcedure in myAdventureWorks.StoredProcedures)

{

StringCollection ProcedureScripts = myProcedure.Script(scriptOptions);

ProcedureScripts = myProcedure.Script();

string newSql = "";

foreach (string script in ProcedureScripts)

{

newSql = newSql + script;

text = text + script;

}

File.WriteAllText(path + @"\Procedures\" + myProcedure.Name + ".sql", newSql);

}

File.WriteAllText(path + @"\" + "AllProcedure.sql", text);

}

**Заключение**

Описанная функциональность может быть полезна для логирования изменений в структуре баз данных, автоматической фильтрации выгружаемых объектов по названию. Например, префикс в названиях может указывать на отдельное направление в проекте. Общие и отдельные скрипты можно складывать в SVN. Можно писать автоматическое сравнение объектов базы и отсылку сообщений ответственному лицу по факту произошедших изменений.