Synchronisation zweier SQL Server Datenbanken

Kathegorie
SQL Server

Aaron
Priesterroth

In unserem heutigen Artikel möchte ich ein Verfahren zur Synchronisation von zwei SQL Server Datenbanken vorstellen und dieses im Anschluss mit der alternativen Verwendung von Automatisierungen für Migration und Synchronisation am Beispiel des SQLSyncers vergleichen.

Problem: Wie vergleicht man Datenbanken Schemata in SQL Server?

Im ersten Abschnitt möchte ich ein Verfahren zum Vergleich von Datenbanken Schemata vorführen. Dabei werden wir ausschließlich auf den Vergleich von Tabellen, Spalten und Constraints (einschließlich der Indexe) eingehen. Andere Objekte, wie beispielsweise Sichten oder gespeicherte Prozeduren, können ebenfalls in das Verfahren einbezogen werden, sind aber nicht Teil des heutigen Artikels.
Um dem Verfahren folgen zu können, empfehlen wir, eine SQL Server 2019 Instanz zu verwenden. Zusätzlich wird die neuste Version von Visual Studio 2019 mit einer .NET Installation der Version 4.8 oder höher vorausgesetzt.

SQL Server Database Schema erzeugen

Um die Struktur zweier SQL Server Datenbanken vergleichen zu können, wird ein Schema zum Vergleich benötigt. Dieses Schema muss also zunächst ausgelesen und anschließend gespeichert werden. Für unseren Vergleich werden wir dabei das Schema im JSON (JavaScript Objekt Notation) Format bereitstellen.

Datenbank-Spalten auslesen

Das erste Puzzlestück zum Erzeugen des Schemas sind die Datenbanken-Spalten, die mit der folgenden Zeichenkette eine Abfrage ausführen kann. Wichtig zu beachten ist hierbei, dass **alle** Spalten ausgelesen werden, auch Spalten die mit Hilfe von **Always Encrypted** verschlüsselt werden.

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
string getColumns =
  "select \n" +
  "   [Db Schema].TABLE_SCHEMA [Table Schema]," +
  "   [Tables].name [Table Name], \n" +
  "   [Table Columns].name [Column Name], \n" +
  "   [Table Columns].is_nullable[is_nullable], \n" +
  "   [Table Columns].is_identity[is_identity], \n" +
  "   [Table Columns].encryption_algorithm_name, \n" +
  "   [Table Columns].encryption_type_desc, \n" +
  "   case when cek.name is null then null else cek.name end as CEK_Name, \n" +
  "   [Column Type].name [data_type], \n" +
  "   cast \n" +
  "       (case when [Column Type].name = 'text' \n" +
  "           then null \n" +
  "           else \n" +
  "               case when [Table Columns].precision = 0 and [Column Type].name
                  <> 'text' \n" +
  "                   then [Table Columns].max_length \n" +
  "                   else null \n" +
  "               end \n" +
  "           end \n" +
  "       as smallint) [max_length], \n" +
  "   cast(case when [Table Columns].precision>0 and [Column Type].precision=
      [Column Type].scale \n" +
  "           then [Table Columns].precision else null end as tinyint)
              [precision], \n" +
  "   cast(case when [Table Columns].precision>0 and [Column Type].precision=
      [Column Type].scale \n" +
  "           then [Table Columns].scale else null end as tinyint) [scale], \n" +
  "   cast(case when [Table Columns].is_identity= 1 \n" +
  "           then seed_value else null end as sql_variant) [seed_value], \n" +
  "   cast(case when [Table Columns].is_identity= 1 \n" +
  "           then increment_value else null end as sql_variant) [increment_
              value], \n" +
  "   cast(case when [Table Columns].default_object_id>0 \n" +
  "           then definition else null end as varchar(4000)) [default_value] \n" +
"from INFORMATION_SCHEMA.TABLES [Db Schema] \n" +
"   join sys.objects [Tables] on [Db Schema].TABLE_SCHEMA = schema_name(
    [Tables].[schema_id]) \n" +
"       and [Db Schema].TABLE_NAME = [Tables].name \n" +
"   join sys.columns [Table Columns] on [Tables].object_id=[Table Columns].
    object_id \n" +
"   left join sys.column_encryption_keys cek on [Table Columns].column_
    encryption_key_id = \n" +
"       CEK.column_encryption_key_id \n" +
"   left join sys.identity_columns id on [Tables].object_id= id.object_id \n" +
"   join sys.types [Column Type] on [Table Columns].system_type_id=[Column
    Type].system_type_id \n" +
"       and [Column Type].system_type_id=[Column Type].user_type_id \n" +
"   left join sys.default_constraints d on [Table Columns].default_object_id=
    d.object_id \n" +
"where [Tables].type= 'u' \n" +
"order by [Table Schema], [Table Name] \n" +
"for json auto, root('DBColumns') \n";
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Fremdschlüssel auslesen

Das zweite Puzzlestück sind die Foreign-Key Constraints. Diese können mit Hilfe der folgenden Zeichenkette ausgelesen werden.
Eine Besonderheit bei dem Auslesen der Fremdschlüssel ist die Tatsache, dass in SQL Server ein Fremdschlüssel in einer Tabelle benutzt werden kann, der wiederum eine andere Tabelle mit unterschiedlichem Schema referenziert. Wir müssen also bei der Abfrage die Informationen bezüglich der Tabellen-Schemata erhalten.

```
string getFkConstraints =
  "select distinct \n" +
  "   ConstraintColumns.TABLE_SCHEMA [Table Schema], \n" +
  "   ConstraintColumns.Table_Name [Table Name], \n" +
  "   [Constraints].CONSTRAINT_NAME [Constraint Name], \n" +
  "   [Constraints].[Referenced Schema], \n" +
  "   [Constraints].[Referenced Table], \n" +
  "   Columns.[Column Name], \n" +
  "   Columns.[Referenced Column], \n" +
  "   Columns.Colfk_id [Column id], \n" +
  "   Columns.RefCol_id [Referenced Column id] \n" +
  "from INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE ConstraintColumns \n" +
  "join \n" +
  "  ( \n" +
  "    select \n" +
  "      tc.TABLE_SCHEMA, \n" +
  "      tc.TABLE_NAME, \n" +
  "      tc.CONSTRAINT_NAME, \n" +
  "      rc.UNIQUE_CONSTRAINT_SCHEMA [Referenced Schema], \n" +
  "      ccolumns.[Referenced Table] \n" +
  "    from INFORMATION_SCHEMA.TABLE_CONSTRAINTS tc \n" +
  "    join INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS rc \n" +
  "      on tc.CONSTRAINT_SCHEMA= rc.CONSTRAINT_SCHEMA and tc.CONSTRAINT_NAME=
      rc.CONSTRAINT_NAME \n" +
  "    join \n" +
  "       ( \n" +
  "           select \n" +
  "               schema_name(fk.schema_id) Table_Schema, \n" +
  "               object_name(fkc.parent_object_id) Table_Name, \n" +
  "               object_name(fkc.constraint_object_id) Constraint_Name, \n" +
  "               object_name(fkc.referenced_object_id) [Referenced Table] \n" +
  "           from sys.foreign_key_columns fkc \n" +
  "           join sys.foreign_keys fk on fk.object_id= fkc.constraint_object_id
          \n" +
  "               and fk.parent_object_id=fkc.parent_object_id \n" +
  "       ) ccolumns \n" +
  "          on tc.TABLE_SCHEMA=ccolumns.TABLE_SCHEMA \n" +
  "              and tc.TABLE_NAME=ccolumns.TABLE_NAME \n" +
  "  ) [Constraints] \n" +
  "  on ConstraintColumns.CONSTRAINT_NAME=[Constraints].CONSTRAINT_NAME \n" +
  "      and ConstraintColumns.TABLE_NAME=[Constraints].TABLE_NAME \n" +
"join \n" +
  "  ( \n" +
  "      select \n" +
  "          schema_name(fk.schema_id) Table_Schema, \n" +
  "          object_name(fkc.parent_object_id) Table_Name, \n" +
  "          object_name(fkc.constraint_object_id) Constraint_Name, \n" +
  "          colfk.name [Column Name], \n" +
  "          object_name(fkc.referenced_object_id) [Referenced Table], \n" +
  "          colref.name [Referenced Column],\n" +
  "          colfk.column_id [Colfk_id],\n" +
  "          colref.column_id [RefCol_id]\n" +
  "      from sys.foreign_key_columns fkc\n" +
  "      join sys.foreign_keys fk on fk.object_id= fkc.constraint_object_id\n" +
  "          and fk.parent_object_id= fkc.parent_object_id\n" +
  "      join sys.columns colfk on fkc.parent_object_id= colfk.object_id\n" +
  "          and fkc.parent_column_id= colfk.column_id\n" +
  "      join sys.columns colref on fkc.referenced_object_id= colref.object_id\n" +
  "          and fkc.referenced_column_id= colref.column_id\n" +
  "  ) Columns \n" +
  "      on ConstraintColumns.TABLE_SCHEMA=Columns.TABLE_SCHEMA \n" +
  "          and ConstraintColumns.TABLE_NAME=Columns.TABLE_NAME \n" +
  "          and ConstraintColumns.CONSTRAINT_NAME=Columns.Constraint_Name\n" +
"order by ConstraintColumns.TABLE_SCHEMA, ConstraintColumns.Table_Name, \n" +
"   [Constraint Name], [Referenced Schema], [Referenced Table], [Column id]\n" +
"for json auto, root('DBFKConstraints') \n";
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Constraints auslesen

Die restlichen Constraints (abgesehen von den Fremdschlüsseln) können mit folgender Abfrage ausgelesen werden:

```
string getConstraints =
  "select distinct \n" +
  "    ConstraintColumns.TABLE_SCHEMA [Table Schema],\n" +
  "    ConstraintColumns.Table_Name [Table Name],\n" +
  "    [Constraints].CONSTRAINT_NAME [Constraint Name],\n" +
  "    [Constraints].type_desc,\n" +
  "    [Constraints].ignore_dup_key,\n" +
  "    [Constraints].CONSTRAINT_TYPE [Constraint Type],\n" +
  "    [Constraints].CHECK_CLAUSE,\n" +
  "    Columns.COLUMN_NAME,\n" +
  "    Columns.is_descending_key,\n" +
  "    Columns.ordinal_position\n" +
  "from INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE ConstraintColumns\n" +
  "join \n" +
  "    (\n" +
  "        select tc.TABLE_SCHEMA, tc.CONSTRAINT_CATALOG, tc.CONSTRAINT_NAME, \n" +
  "            tc.CONSTRAINT_TYPE, tc.TABLE_NAME, ck.CHECK_CLAUSE, \n" +
  "            i.type_desc, I.ignore_dup_key\n" +
  "        from INFORMATION_SCHEMA.TABLE_CONSTRAINTS tc\n" +
  "        left join sys.indexes i on OBJECT_name(i.object_id) = tc.TABLE_NAME\n" +
  "            and i.name = tc.CONSTRAINT_NAME\n" +
  "        left join INFORMATION_SCHEMA.CHECK_CONSTRAINTS ck \n" +
  "            on tc.CONSTRAINT_CATALOG=ck.CONSTRAINT_CATALOG\n" +
  "                and tc.CONSTRAINT_NAME=ck.CONSTRAINT_NAME\n" +
  "    ) [Constraints] \n" +
  "        on ConstraintColumns.CONSTRAINT_NAME=[Constraints].CONSTRAINT_NAME \n" +
  "            and ConstraintColumns.CONSTRAINT_CATALOG=[Constraints].CONSTRAINT_
            CATALOG\n" +
  "            and ConstraintColumns.TABLE_NAME=[Constraints].TABLE_NAME\n" +
  "join \n" +
  "    (\n" +
  "        select CheckColumns.TABLE_SCHEMA,\n" +
  "            CheckColumns.TABLE_NAME, \n" +
  "            CheckColumns.COLUMN_NAME, \n" +
  "            CheckColumns.CONSTRAINT_NAME, \n" +
  "            null as is_descending_key,\n" +
  "            null as ordinal_position\n" +
  "        from INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE CheckColumns \n" +
  "        where CheckColumns.CONSTRAINT_NAME in \n" +
  "            (\n" +
  "                select CONSTRAINT_NAME from INFORMATION_SCHEMA.CHECK_
                    CONSTRAINTS\n" +
  "            )\n" +
  "        union all\n" +
  "        select \n" +
  "            ConstraintColumns.TABLE_SCHEMA,\n" +
  "            ConstraintColumns.TABLE_NAME, \n" +
  "            ConstraintColumns.COLUMN_NAME, \n" +
  "            ConstraintColumns.CONSTRAINT_NAME, \n" +
  "            ConstraintColumns.is_descending_key,\n" +
  "            ConstraintColumns.ordinal_position\n" +
  "        from \n" +
  "            (\n" +
  "                select\n" +
  "                    schema_name(obj.schema_id) TABLE_SCHEMA,\n" +
  "                    obj.name TABLE_NAME,\n" +
  "                    i.name CONSTRAINT_NAME,\n" +
  "                    index_column_id ordinal_position,\n" +
  "                    (\n" +
  "                        select name from sys.columns cols \n" +
  "                        where cols.object_id=obj.object_id and ic.column_id=c
                        ols.column_id\n" +
  "                    ) column_name,\n" +
  "                    is_descending_key\n" +
  "                from sys.indexes i\n" +
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
  "                   join sys.objects obj on obj.object_id=i.object_id and obj.
                      type='u'\n" +
  "                   join sys.index_columns ic on obj.object_id=ic.object_id\n" +
  "                       and ic.index_id=i.index_id\n" +
  "                where is_primary_key=1 or is_unique_constraint=1\n" +
  "            ) ConstraintColumns\n" +
  "    ) Columns \n" +
  "        on ConstraintColumns.TABLE_SCHEMA=Columns.TABLE_SCHEMA\n" +
  "            and ConstraintColumns.TABLE_NAME=Columns.TABLE_NAME \n" +
  "            and ConstraintColumns.CONSTRAINT_NAME=Columns.CONSTRAINT_NAME\n" +
  "order by \n" +
  "    ConstraintColumns.TABLE_SCHEMA, \n" +
  "    ConstraintColumns.Table_Name, \n" +
  "    [Constraint Name], \n" +
  "    Columns.ordinal_position\n" +
  "for json auto, root('DBConstraints')";
```

Indexe auslesen

Das letzte Puzzlestück sind die Tabellen-Indexe. Sie können mit der folgenden Abfrage ausgelesen werden:

```
string getIndexes =
  "select \n" +
  "   schema_name(obj.schema_id) [Table Schema], \n" +
  "   obj.name [Table Name], \n" +
  "   [Indexes].name index_name, \n" +
  "   [Indexes].type_desc, \n" +
  "   index_column_id [column_id], \n" +
  "   ( \n" +
  "       select name from sys.columns cols \n" +
  "       where cols.object_id=obj.object_id and Columns.column_id= cols.column_
        id \n" +
  "   ) column_name, \n" +
  "   ignore_dup_key, \n" +
  "   is_descending_key \n" +
  "from sys.indexes [Indexes] \n" +
  "   join sys.objects obj on obj.object_id=[Indexes].object_id and obj.type= 'u'
    \n" +
  "   join sys.index_columns Columns on obj.object_id= Columns.object_id \n" +
  "       and Columns.index_id=[Indexes].index_id \n" +
  "where is_primary_key=0 and is_unique_constraint=0 \n" +
  "order by [Table Schema], [Table Name], index_name, column_id \n" +
  "for json auto, root('DBIndexes') \n";
```

Datenbank Schema als JSON-Datei

Die vier Abfragen bezüglich der Spalten, Fremdschlüssel, Constraints und Indexe können nun dazu verwendet werden, das Schema einer Datenbank zu auszulesen. Die Ausgaben der Abfragen sollen dabei in einer Datei im JSON-Format gespeichert werden.

```
private void GetColumns(SqlCommand cmd)
  {
    _columnsFromDb.Clear();
    cmd.CommandText = getColumns;
    try
    {
      SqlDataReader reader = cmd.ExecuteReader();
      StringBuilder sb = new StringBuilder();
      // Need this cycle to read the full content, otherwise it gets truncated!!!
      while(reader.Read())
      {
        sb.Append(reader.GetSqlString(0).Value);
      }
      reader.Close();
      string json = sb.ToString();
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
      dynamic cols = JsonConvert.DeserializeObject(json);

      _columnsFromDb = cols == null ? new JArray() : (JArray)cols["DBColumns"];
      // If we have an empty database, we don't have columns
      if(_columnsFromDb.Count == 0)
      {
         _error += "\r\nMissing columns from schema file";
         _haveErrors = true;
      }
    }
    catch(Exception e)
    {
      _error += "\r\nError reading database: " + e.Message + ")";
      _haveErrors = true;
    }
  }
}
```

Achtung: Das Schema einer Datenbank kann größer sein als die maximale Größe einer Rückgabe die vom **SqlDataReader** mit einem Aufruf zurückgegeben werden kann. Wir verwenden deshalb eine **while**-Schleife, um die Rückgabe stückweise auszulesen.

Alle Fehlermeldungen, die während der Ausführung vom SQL Server geworfen werden, werden am Ende des Programms an den Benutzer ausgegeben.

Falls gewünscht, kann eine einfache Verschlüsselung zum Speichern des ausgelesenen Schemas verwendet werden, um es vor unbefugten Blicken zu schützen. **base64** bietet sich beispielsweise an dieser Stelle besonders an.
Falls stattdessen gewünscht ist, dass das ausgelesene Schema nachträglich nicht unbemerkt verändert werden kann, bietet sich die Verwendung einer Prüfsumme in der ersten Zeile der JSON-Datei an. Im folgenden Code-Beispiel werden wir eine **base64** Repräsentation eines SHA512-Hashes der JSON-Datei als Prüfsumme verwenden.

```
dynamic dbschema = new JObject();
try
{
    _conn.ChangeDatabase(_dbName);
    GetDatabaseElements();
    dbschema.DBColumns = _columnsFromDb;
    dbschema.DBFKConstraints = _fkConstraintsFromDb;
    dbschema.DBConstraints = _constraintsFromDb;
    dbschema.DBIndexes = _indexesFromDb;

    // Delete the previous database structure file
    File.Delete(_jsonFile);
    string jsonstr = dbschema.ToString() + "\n";
    byte[] data = Encoding.UTF8.GetBytes(jsonstr);
    string json64 = Convert.ToBase64String(data);

    // Make sure the database structure hasn't been tampered
    using SHA512 sha512Hash = SHA512.Create();
    byte[] tempSha512;
    tempSha512 = sha512Hash.ComputeHash(data);
    string hash = Convert.ToBase64String(tempSha512) + "\n";
    using StreamWriter jsonout = new StreamWriter(_jsonFile);
    if(!_ignoreChecksum)
        jsonout.Write(hash);
    jsonout.Write(_encryptSchema ? json64 : jsonstr);
}
catch(Exception e)
{
    _error += "\r\nError saving database schema: " + e.Message + "\n";
    _haveErrors = true;
}
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Datenbank von Schema erstellen

Um ein Gefühl für die erstellten Schemata zu bekommen, betrachten wir einmal das Schema einer Spalte, die mit Hilfe von **Always Encrypted** verschlüsselt ist.

```
"Table Schema": "dbo",
"Tables": [
  {
    "Table Name": "MyDemoTable",
    "Table Columns": [
      {
        "Column Name": "SecureInformation",
        "is_nullable": false,
        "is_identity": false,
        "encryption_algorithm_name": "AEAD_AES_256_CBC_HMAC_SHA_256",
        "encryption_type_desc": "DETERMINISTIC",
        "CEK_Name": "CEK_MyAlwaysEncryptedTest",
        "Column Type": [
          {
            "data_type": "varchar",
            "max_length": 34
          }
        ]
      }
    ]
  }
]
```

Möchten wir eine verschlüsselte Spalte in der neuen Datenbank anlegen, müssen wir lediglich die folgenden drei Zeilen an das Schema der Datenbank anfügen:

```
"encryption_algorithm_name": "AEAD_AES_256_CBC_HMAC_SHA_256",
"encryption_type_desc": "DETERMINISTIC",
"CEK_Name": "CEK_MyAlwaysEncryptedTest",
```

Hinweis: An dieser Stelle ist es wichtig anzumerken, dass wir den CEK als Schlüssel in der Konfigurationsdatei des Programms verwenden. Der CEK, der eine spezifische Spalte identifiziert, wird dabei ignoriert. Wir gehen also davon aus, dass sie den selben Schlüssel verwenden.

Datenbank Schema aus JSON-Datei auslesen

Mithilfe des folgenden Codes kann das Datenbank Schema von der zuvor erstellten JSON-Datei ausgelesen werden:

```
private void ReadDBStructure()
{
  string jsonstr = string.Empty;
  string hashstr = string.Empty;
  try
  {
      using(StreamReader reader = new StreamReader(_jsonFile))
      {
          if(!_ignoreChecksum) // get the hash which is in first line
              hashstr = reader.ReadLine();
          if(_encryptSchema)
          {
              string tmpstr = reader.ReadToEnd();
              byte[] data = Convert.FromBase64String(tmpstr);
              jsonstr = Encoding.ASCII.GetString(data);
          }
          else
          {
              jsonstr = reader.ReadToEnd();
          }
      }
      if(!_ignoreChecksum)
      {
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
    byte[] jsondata = Encoding.UTF8.GetBytes(jsonstr);

    // Make sure the database structure hasn't been tampered
    using SHA512 sha512Hash = SHA512.Create();
    byte[] temphash512;
    temphash512 = sha512Hash.ComputeHash(jsondata);
    string hashjson = Convert.ToBase64String(temphash512);
    if(hashstr != hashjson) // Database structure has been tampered!
    {
        _error += "\r\nDatabase structure has been tampered!";
        _haveErrors = true;
        return;
    }
}

// When saved from SSMS the json string is encoded with html tags, like
   &gt; &lt; etc
// We don't need them
jsonstr = HttpUtility.HtmlDecode(jsonstr);
// Get the database json structure
dynamic array = JsonConvert.DeserializeObject(jsonstr);
_columnsFromJson = array["DBColumns"];
_fkConstraintsFromJson = array["DBFKConstraints"];
if(_fkConstraintsFromJson == null)
{
    _fkConstraintsFromJson = new JArray();
}
_constraintsFromJson = array["DBConstraints"];
if(_constraintsFromJson == null)
{
    _constraintsFromJson = new JArray();
}
_indexesFromJson = array["DBIndexes"];
if(_indexesFromJson == null)
{
    _indexesFromJson = new JArray();
    }
    GetTablesFromJson();
  }
  catch(Exception e)
  {
    _error += "\r\nError parsing database structure file: " + e.Message;
    _haveErrors = true;
  }
}
```

Wie vielleicht an dieser Stelle auffällt, beinhaltet das erstellte Schema keine Tabellen, nur die Spalten. Möchten wir eine neue Datenbank mit Hilfe des ausgelesenen Schemas erstellen, müssen wir diese reproduzieren.
Was wir wissen ist, dass jede Spalte zu einer Tabelle gehört. Damit wissen wir auch, dass alle Tabellen, die wir benötigen, ebenfalls im Schema enthalten sind. Nur eben nicht in einem Format in dem wir es zur Wiederherstellung der Datenbank gebrauchen können. Aus diesem Grund verwenden wir die Funktion **GetTablesFromJson()**, um alle Tabellen aus den Informationen der Spalten "zu sammeln".

```
private void GetTablesFromJson()
{
    // We have no columns, so we have no tables
    _tablesFromJson = _columnsFromJson.Count == 0
        ? (dynamic)new ArrayList()
        : (dynamic)(from schema in _columnsFromJson
                    select new
                    {
                        schemaName = (string)schema["Table Schema"],
                        tables = (from t in schema["Tables"]
                                  select (string)t["Table Name"]).ToList()
                    }).ToList();
}
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Datenbank Tabellen erstellen

Nachdem die Tabellen-Struktur reproduziert wurde, muss sie nur noch erstellt werden. Dabei hilft der folgende Code:

```
private void CreateTables(SqlCommand cmd)
{
  string query = string.Empty;
  try
  {
    foreach(JToken schema in _columnsFromJson)
    {
      // Table schema might be missing from Json file
      string schemaName = (string)schema["Table Schema"];
      if(string.IsNullOrWhiteSpace(schemaName))
          schemaName = "dbo";

      // Create schema if it doesn't exist
      query = "IF(NOT EXISTS(SELECT * FROM sys.schemas WHERE name = '" +
      schemaName + "'))\n" +
              "BEGIN\n" +
              "    EXEC('CREATE SCHEMA [" + schemaName + "] AUTHORIZATION [dbo]')
                   \n" +
              "END;";
      cmd.CommandText = query;
      cmd.ExecuteNonQuery();

      JArray tablesFromSchema = (JArray)schema["Tables"];
      foreach(var (table, fullTableName) in from JToken table in tablesFromSchema
                  let tableName = "[" + (string)table["Table Name"] + "]"
                  let fullTableName = "[" + schemaName + "]." + tableName
                  select (table, fullTableName))
      {
        // Start building the query for creating the table
        query = "CREATE TABLE " + fullTableName + " (\n";

        JArray tableColumns = (JArray)table["Table Columns"];
        foreach(JToken column in tableColumns)
        {
            string columnName, datatype = string.Empty;
            string encryptionType, encrypttext = string.Empty;
            string defaultValue = null;
            bool isNullable = false, isIdentity = false;
            int? maxlen = null;
            int seed = 0, increment = 0;
            int? precision = null, scale = null;

            columnName = (string)column["Column Name"];
            isNullable = (bool)column["is_nullable"];
            isIdentity = (bool)column["is_identity"];
            encryptionType = (string)column["encryption_type_desc"];
            //cekName = (string)column["CEK_Name"];
            if(!string.IsNullOrWhiteSpace(encryptionType) && !string.
            IsNullOrWhiteSpace(_cek))
              encrypttext = " COLLATE Latin1_General_BIN2 ENCRYPTED WITH (COLUMN_
              ENCRYPTION_KEY=["+
                 _cek + "], ENCRYPTION_TYPE = " + encryptionType +
                 ", ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256')";

            JArray columnType = (JArray)column["Column Type"];
            foreach(JToken coltype in columnType)
            {
                datatype = (string)coltype["data_type"];
                defaultValue = (string)coltype["default_value"];
                maxlen = (int?)coltype["max_length"];
                if(isIdentity)
                {
                    seed = (int)coltype["seed_value"];
                    increment = (int)coltype["increment_value"];
                }
                precision = (int?)coltype["precision"];
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
    scale = (int?)coltype["scale"];
}
if(!string.IsNullOrWhiteSpace(datatype))
    query += "    [" + columnName + "] " + datatype;
else // We should NOT get here!!!
{
    _error += "\r\nError in database schema, in table " +
        fullTableName + ", column [" + columnName +
        "] has no data type";
    _haveErrors = true;
    continue; // go to next column, trying to collect all errors
}

if(precision.HasValue)
{
    query += "(" + precision.Value;
    if(scale.HasValue)
        query += ", " + scale.Value;
    query += ") ";
}
else if(maxlen.HasValue)
    query += "(" + maxlen.Value + ") ";
if(isIdentity)
{
    query += " identity(" + seed + ", " + increment + ") NOT NULL";
}
else
{
    if(!string.IsNullOrWhiteSpace(encrypttext))
        query += encrypttext;
    if(!isNullable)
    {
        query += " NOT NULL ";
    }
}
                    if(!string.IsNullOrWhiteSpace(defaultValue))
                        query += " DEFAULT " + defaultValue;
                    query += ",\n";
                }
                // Replace the last , with )
                query = query.Substring(0, query.LastIndexOf(',')) + "\n)";
                // Create the table
                cmd.CommandText = query;
                cmd.ExecuteNonQuery();
                _diffmsg += "\r\n\tCreate table " + fullTableName + "...";
            }
        }
    }
    catch(Exception e)
    {
        _error += "\r\nError creating  tables: " + e.Message;
        _haveErrors = true;
    }
}
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Indexe aufbauen

Nachdem die Tabellen erstellt wurden, können wir mit dem Aufbau der Indexe beginnen.
Die benötigten Informationen sind im zuvor erstellten Schema vorhanden.
Mit folgendem Code können die Indexe erstellt werden:

```
private void CreateIndexes(SqlCommand cmd)
{
    string query = string.Empty;
    try
    {
        foreach(var (fullTableName, arrayjs) in
            from JToken table in _indexesFromJson
            let schemaName = (string)table["Table Schema"]
            let tableName = (string)table["Table Name"]
            let fullTableName = "[" + schemaName + "].[" + tableName + "]"
            let arrayjs = (JArray)table["Indexes"]
            select (fullTableName, arrayjs))
        {
            foreach(var (index, indexname, columns, typedesc, ignoreDupKey) in
                from JToken index in arrayjs
                let indexname = "[" + (string)index["index_name"] + "]"
                let columns = (JArray)index["Columns"]
                let typedesc = (string)index["type_desc"]
                let ignoreDupKey = (bool)index["ignore_dup_key"]
                select (index, indexname, columns, typedesc, ignoreDupKey))
            {
                query = "CREATE " + typedesc + " INDEX " + indexname + " ON "
                    + fullTableName + "\n(\n";
                foreach(var column in columns)
                {
                    bool isDescending = (bool)column["is_descending_key"];
                    query += "    [" + (string)column["column_name"] + "] ";
                    if(isDescending)
                        query += " DESC";
                    query += ",\n";
                }
                // Replace the last , with )
                query = query.Substring(0, query.LastIndexOf(',')) + "\n)";
                if(ignoreDupKey)
                    query += " WITH (IGNORE_DUP_KEY = ON)\n";
                cmd.CommandText = query;
                cmd.ExecuteNonQuery();
                _diffmsg += "\r\n\tCreate index [" + indexname + "] for table "
                    + fullTableName + "...";
            }
        }
    }
    catch(Exception e)
    {
        _error += "\r\nError creating indexes: " + e.Message;
        _haveErrors = true;
    }
}
```

Constraints wiederherstellen

Nach den Indexen können nun mit folgendem Code die Constraints hergestellt werden:

```
private void CreateNonFkConstraints(SqlCommand cmd)
{
  // Create Primary Keys, Unique and Check constraints
  string query = string.Empty;
  try
  {
    foreach(var (schemaName, tableName, fullTableName, arrayjs) in
        from JToken table in _constraintsFromJson
        let schemaName = (string)table["Table Schema"]
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
  let tableName = (string)table["Table Name"]
  let fullTableName = "[" + schemaName + "].[" + tableName + "]"
  let arrayjs = (JArray)table["Constraints"]
  select (schemaName, tableName, fullTableName, arrayjs))
{
  foreach(var (constraintname,constrainttype,checkclause,columns,typedesc,
  ignoreDupKey) in
      from JToken constraint in arrayjs
      let constraintname = "[" + (string)constraint["Constraint Name"] + "]"
      let constrainttype = (string)constraint["Constraint Type"]
      let checkclause = (string)constraint["CHECK_CLAUSE"]
      let columns = (JArray)constraint["Columns"]
      let typedesc = (string)constraint["type_desc"]
      let ignoreDupKey = (bool?)constraint["ignore_dup_key"]
      select (constraintname, constrainttype, checkclause, columns,
      typedesc, ignoreDupKey))
      {
          query = "ALTER TABLE " + fullTableName;
          if(constrainttype == "PRIMARY KEY" || constrainttype == "UNIQUE")
          {
              query += " ADD Constraint " + constraintname + " " +
              constrainttype + " " +
                  typedesc + "\n(\n";
              foreach(var column in columns)
              {
                  bool isDescending = (bool)column["is_descending_key"];
                  query += "    [" + (string)column["COLUMN_NAME"] + "] ";
                  if(isDescending)
                      query += " DESC";
                  query += ",\n";
              }
              // Replace the last , with )
              query = query.Substring(0, query.LastIndexOf(',')) + "\n)";
              if(ignoreDupKey.HasValue && ignoreDupKey.Value)
                  query += " WITH (IGNORE_DUP_KEY = ON)\n";
              }
              else if(constrainttype == "CHECK")
              {
                  query += " WITH CHECK Add constraint [" + constraintname + "]
                  " +
                      constrainttype + " " + checkclause;
                  query += ";\nALTER TABLE " + fullTableName + "Check
                  Constraint [" +
                      constraintname + "];\n";
              }
              else
              {
                 _error += "\r\nError creating database constraints: Unknown
                 constraint type [" +
                      constrainttype + "]";
                  _haveErrors = true;
                  continue;
              }
              cmd.CommandText = query;
              cmd.ExecuteNonQuery();
              _diffmsg += "\r\n\tAdd constraint " + constraintname + " to table
              " +
                  fullTableName + "...";
          }
      }
  }
  catch(Exception e)
  {
      _error += "\r\nError creating constraints: " + e.Message;
      _haveErrors = true;
  }
}
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Fremdschlüssel erzeugen

Zu guter Letzt müssen jetzt nur noch die Fremdschlüssel erzeugt werden. Anschließend ist die Datenbank, basierend auf dem ausgelesenen Schema, wiederhergestellt.

```
private void CreateFkConstraints(SqlCommand cmd)
{
    // Create Foreign Keys
    string query = string.Empty;
    try
    {
        foreach(var (schemaName, tableName, fullTableName, arrayjs) in
            from JToken table in _fkConstraintsFromJson
            let schemaName = (string)table["Table Schema"]
            let tableName = (string)table["Table Name"]
            let fullTableName = "[" + schemaName + "].[" + tableName + "]"
            let arrayjs = (JArray)table["Constraints"]
            select (schemaName, tableName, fullTableName, arrayjs))
        {
            foreach(var (constraintname, referencedTable, columns) in
                from JToken constraint in arrayjs
                let constraintname = "[" + (string)constraint["Constraint Name"]
                + "]"
                let refschema = (string)constraint["Referenced Schema"]
                let reftable = (string)constraint["Referenced Table"]
                let referencedTable = "[" + refschema + "].[" + reftable + "]"
                let columns = (JArray)constraint["Columns"]
                select (constraintname, referencedTable, columns))
            {
                query = "ALTER TABLE " + fullTableName + " WITH CHECK Add
                constraint ";
                query += constraintname + " Foreign Key(";
                foreach(var column in columns)
                {
                    query += "[" + (string)column["Column Name"] + "], ";
                }
                // Replace the last , with )
                query = query.Substring(0, query.LastIndexOf(',')) + ")\n";
                query += "    references " + referencedTable + " (";
                foreach(var column in columns)
                {
                    query += "[" + (string)column["Referenced Column"] + "], ";
                }
                // Replace the last , with )
                query = query.Substring(0, query.LastIndexOf(',')) + ");\n";
                query += "ALTER TABLE " + fullTableName + " CHECK constraint [" +
                    constraintname + "];";
                cmd.CommandText = query;
                cmd.ExecuteNonQuery();
                _diffmsg += "\r\n\tAdd FOREIGN KEY " + constraintname + " to
                table " +
                    fullTableName + "...";
            }
        }
    }
    catch(Exception e)
    {
        _error += "\r\nError creating FOREIGN KEYs: " + e.Message;
        _haveErrors = true;
    }
}
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Die erzeugten Schemata vergleichen

Nachdem wir nun die Schemata sowie die Reproduktion der Datenbanken erzeugt und gezeigt haben, werden nun die beiden Schemata miteinander verglichen.

Tabellen vergleichen

Bei dem Vergleich der Tabellen beschränken wir uns lediglich auf die Betrachtung des Schemas: finden wir im Schema eine Tabelle, die in der neuen Datenbank nicht existiert, speichern wir die Struktur der Tabelle und erstellen sie anschließend.

```
foreach(var schema in _tablesFromJson)
{
  foreach(var tableName in schema.tables)
  {
    string schemaName = schema.schemaName;
    if(!FoundTable(schemaName, tableName, _tablesFromDb))
    {
      // table not found, get its columns and create it
      string fullTableName = "[" + schemaName + "].[" + tableName + "]";
      _diffmsg += "\r\nTable " + fullTableName + " is missing from database";

      //missingElementsFromDb[colnamejs] = tableNameJson;
      _haveDiff = true;

      foreach(var (schemaNameJson, tableJson) in
        from JToken tokenJson in _columnsFromJson
        let schemaNameJson = (string)tokenJson["Table Schema"]
        let tableJson = (JArray)tokenJson["Tables"]
        where schemaNameJson == schemaName //&& tableJson["Table Name"] ==
        tableName
        select (schemaNameJson, tableJson))
      {
        foreach(var (table, tableNameJson) in
          from JToken tokenJsonTbl in tableJson
          let table = (JArray)tokenJsonTbl["Table Columns"]
          let tableNameJson = (string)tokenJsonTbl["Table Name"]
          where schemaNameJson == schemaName //&& tableJson["Table Name"] ==
          tableName
          select (table, tableNameJson))
        {
          _extraTablesInJson[fullTableName] = table;
        }
      }
    }
  }
}
```

Spalten vergleichen

Für den Vergleich der Spalten empfehlen wir ein anderes Vorgehen: für jede Spalte, die wir im Schema finden, müssen wir überprüfen, ob diese Spalte in der neuen Datenbank existiert. Gleichzeitig müssen wir aber auch für jede Spalte in der Datenbank überprüfen, ob diese im Schema existiert.
Dafür benutzen wir die Funktion mit dem Namen CompareColumns(), eine mögliche Implementation könnte wie folgt aussehen:

```
foreach(JToken schemaDb in _columnsFromDb)
{
    // Table schema should NEVER be null! It always contains at least "dbo" as
       schema name
    string schemaNameDb = (string)schemaDb["Table Schema"];
    JArray tablesFromSchemaDB = (JArray)schemaDb["Tables"]; // Get the tables in
    each schema

    foreach(JToken schemaJson in _columnsFromJson)
    {
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
// Table schema might be missing from Json file
string schemaNameJson = (string)schemaJson["Table Schema"];
if(schemaNameDb != schemaNameJson) // Different schema
    continue;
JArray tablesFromSchemaJson = (JArray)schemaJson["Tables"];
foreach(JToken tableDb in tablesFromSchemaDB)
{
    string tableNameDb = (string)tableDb["Table Name"];
    string fullTableName = "[" + schemaNameDb + "].[" + tableNameDb +
    "]";

    if(!FoundTable(schemaNameDb, tableNameDb, _tablesFromJson))
    {
        _diffmsg += "\r\nFound extra table " + fullTableName + " in
        database";
        _haveDiff = true;
        continue;
    }

    JArray arraydb = (JArray)tableDb["Table Columns"];
    foreach(var (tableJson, tableNameJson) in
        from JToken tableJson in tablesFromSchemaJson
        let tableNameJson = (string)tableJson["Table Name"]
        select (tableJson, tableNameJson))
    {
        // We search if this table from db structure is in database,
        // otherwise we need to create it
        string jsonTableName = "[" + schemaNameJson + "].[" +
        tableNameJson + "]";
        if(!FoundTable(schemaNameDb, tableNameDb, _tablesFromDb))
        {
            _diffmsg += "\r\nMissing table " + jsonTableName + " from
            database";
            _haveDiff = true;
            continue;
        }
                if(tableNameDb != tableNameJson) // different table
                    continue;
                JArray arrayjs = (JArray)tableJson["Table Columns"];
                CompareColumnsFromTable(fullTableName, arraydb, arrayjs);
            }
        } // foreach(JToken tableDb in tablesFromSchemaDB)
    } // foreach(JToken schemaJson in _columnsFromJson)
} // foreach(JToken schemaDb in _columnsFromDb)
```

Die Funktion CompareColumns() benötigt zwei weitere Hilfsfunktionen. Zum einen eine Funktion, die überprüfen kann, ob eine Spalte in der JSON-Datei (dem Schema) existiert:

```
private bool FoundTable(string schemaName, string tableName, dynamic where)
{
    return ((IEnumerable)where).Cast<dynamic>()
        .Where(s => s.schemaName.ToString() == schemaName)
        .Select(t => t.tables).ToList().First().Contains(tableName);
}
```

Außerdem benötigen wir eine Funktion, die die strukturelle Beschaffenheit von zwei Spalten vergleicht:

```
private void CompareColumnsFromTable(string tableName, JArray tableDb, JArray
tableJso)
{
    JObject missingColumnsFromDb = new JObject();
    JObject missingColumnsFromJson = new JObject();
    JObject differentColumns = new JObject();

    // Compare each column of database with each column of schema
    foreach(var (coldb, colnamedb) in from JToken coldb in tableDb
                                      let colnamedb = (string)coldb["Column
Name"]
                                      select (coldb, colnamedb))
    {
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
    bool found = false;
    foreach(var coljs in from JToken coljs in tableJson
                         let colnamejs = (string)coljs["Column Name"]
                         where colnamejs != null && colnamedb == colnamejs
                         select coljs)
    {
        found = true;
        if(!JToken.DeepEquals(coldb, coljs)) // we have differences
        {
            _diffmsg += "\r\nIn table " + tableName + " column [" +
            colnamedb + "] is different in database";

            // Nqw we store the full columns definition
            // We need this when deciding if column change is possible
            JObject differences = new JObject
            {
                ["database"] = coldb,
                ["json"] = coljs
            };
            _haveDiff = true;
            break;
        }
    }
    if(!found)
    {
        _diffmsg += "\r\nIn table " + tableName + " column [" +
            colnamedb + "] is missing from schema file";
        _haveDiff = true;
    }
}

// Compare each column of schema with each column of database
foreach(var (coljs, colnamejs) in from JToken coljs in tableJson
                                  let colnamejs = (string)coljs["Column Name"]
                                  select (coljs, colnamejs))
{
    bool found = false;

    // Here we look ONLY for columns in json file which are NOT in the database
    // These columns need to be added in the database
    foreach(var _ in from JToken coldb in tableDb
                     let colnamedb = (string)coldb["Column Name"]
                     where colnamejs == colnamedb
                     select new
                     {
                     })
    {
        found = true;
        break;
    }

    if(!found)
    {
        _diffmsg += "\r\nIn table " + tableName + " column [" +
            colnamejs + "] is missing from database";
        _haveDiff = true;
    }
  }
}
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Indexe vergleichen

Nachdem die Spalten abgeglichen wurden, können nun zu guter Letzt die Indexe verglichen werden. Dies geschieht wieder in zwei getrennten Schritten.

Im ersten Schritt muss überprüft werden, ob jedes der in der JSON-Datei befindlichen Elemente auch in der Datenbank zu finden ist (selber Name, die selben Spalten und Reihenfolge der Spalten). Das geschieht mit folgendem Code:

```
foreach(var (schemaNameJson, tableNameJson, fullJsonTableName, arrayjs) in
    from JToken tableJson in _indexesFromJson
    let schemaNameJson = (string)tableJson["Table Schema"]
    let tableNameJson = (string)tableJson["Table Name"]
    let fullJsonTableName = "[" + schemaNameJson + "].[" + tableNameJson + "]"
    let arrayjs = (JArray)tableJson["Indexes"]
    select (schemaNameJson, tableNameJson, fullJsonTableName, arrayjs))
{
    foreach(var (coljs, colnamejs) in
        from JToken coljs in arrayjs
        let colnamejs = (string)coljs["index_name"]
        select (coljs, colnamejs))
    {
        bool found = false;
        foreach(var (tableNameDb, fullDbTableName, arraydb) in
            from JToken tableDb in _indexesFromDb
            let schemaNameDb = (string)tableDb["Table Schema"]
            let tableNameDb = (string)tableDb["Table Name"]
            let fullDbTableName = "[" + schemaNameDb + "].[" + tableNameDb + "]"
            let arraydb = (JArray)tableDb["Indexes"]
            select (tableNameDb, fullDbTableName, arraydb))
        {
            // Compare each index from database with each index from schema
            foreach(var (coldb, colnamedb) in
                from JToken coldb in arraydb
                let colnamedb = (string)coldb["index_name"]
                where fullJsonTableName == fullDbTableName && colnamejs == colnamedb
                select (coldb, colnamedb))
            {
                if(!JToken.DeepEquals(coldb, coljs)) // we have differences
                {
                    _diffmsg += "\r\nIn table " + fullJsonTableName + " index ["
+
                        colnamejs + "] is different in database";
                    _haveDiff = true;
                }
                found = true;
            }
            if(found) // exit this cycle as well, we found the same index
                break;
        }
        if(!found)
        {
            _diffmsg += "\r\nIndex [" + colnamejs + "] is missing from database,
table " +
                fullJsonTableName;
            _extraIndexesInJson.Add
            (new JObject
            {
                ["Table Name"] = fullJsonTableName,
                ["Index"] = coljs.DeepClone()
            }
            );
            _haveDiff = true;
        }
    }
}
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Im zweiten Schritt wird sichergestellt, dass für jedes Element in der Datenbank ein äquivalentes Element im Schema (also in der JSON-Datei) zu finden ist:

```
foreach(var (schemaNameDb, tableNameDb, fullDbTableName, arraydb) in
    from JToken tableDb in _indexesFromDb
    let schemaNameDb = (string)tableDb["Table Schema"]
    let tableNameDb = (string)tableDb["Table Name"]
    let fullDbTableName = "[" + schemaNameDb + "].[" + tableNameDb + "]"
    let arraydb = (JArray)tableDb["Indexes"]
    select (schemaNameDb, tableNameDb, fullDbTableName, arraydb))
{
    foreach(var (coldb, colnamedb) in
        from JToken coldb in arraydb
        let colnamedb = (string)coldb["index_name"]
        select (coldb, colnamedb))
    {
        bool found = false;
        foreach(var (schemaNameJson, tableNameJson, fullJsonTableName, arrayjs)
in
            from JToken tableJson in _indexesFromJson
            let schemaNameJson = (string)tableJson["Table Schema"]
            let tableNameJson = (string)tableJson["Table Name"]
            let fullJsonTableName = "[" + schemaNameJson + "].[" + tableNameJson
+ "]"
            let arrayjs = (JArray)tableJson["Indexes"]
            select (schemaNameJson, tableNameJson, fullJsonTableName, arrayjs))
        {
            // Compare each index from database with each index from schema
            foreach(var colnamejs in
                from JToken coljs in arrayjs
                let colnamejs = (string)coljs["index_name"]
                where fullJsonTableName == fullDbTableName && colnamejs ==
colnamedb
                select colnamejs)
            {
                found = true;
                break;
            }
            if(found)
                break;
        }
        if(!found)
        {
            _diffmsg += "\r\nIndex [" + colnamedb + "] is missing from schema
file, table " +
                fullDbTableName;
            _haveDiff = true;
        }
    }
}
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

Synchronisation von Änderungen an der Datenbank

Im letzten Schritt betrachten wir nun, wie Änderungen zwischen den Datenbanken umgesetzt werden können. Um ein einheitliches Verständnis von den unterschiedlichen Situationen zu bekommen, müssen wir zunächst einige Begrifflichkeiten definieren:

- Ist die Rede von **Tabellen/Spalten/Constraints/Index** sprechen wir von Tabellen/Spalten/Constraints/Index, die auf beiden Datenbanken existieren (also Datenbank und JSON-Datei).
- Reden wir von einer **fehlenden Tabelle/Spalte/Constraint/Index** beschreiben wir die Situation, in der eine Tabelle/Spalte/Constraint/Index in dem erzeugten Schema zwar existiert, aber nicht in der Datenbank.
- Reden wir von einer **zusätzlichen Tabelle/Spalte/Constraint/Index** beschreiben wir die Situation in der eine Tabelle/Spalte/Constraint/Index nur in der Datenbank, aber nicht in dem bereitgestellten Schema existiert.
- Sprechen wir in den folgenden Abschnitten von einem **Schema**, meinen wir damit ein bereitgestelltes Schema im JSON-Format, das dazu genutzt wird, Änderungen an der ursprünglichen Datenbank auf der neuen Datenbank umzusetzen.

Möchten wir nun also beispielsweise Elemente von der Datenbank entfernen, müssen wir auf eine bestimmte Reihenfolge der Operationen die auszuführen sind achten: stellen wir uns vor wir haben eine zusätzliche Tabelle mit einem primären Schlüssel und einem Fremdschlüssel in einer der Tabellen. Möchten wir die Tabelle entfernen, ist dies erst möglich nachdem der Fremdschlüssel entfernt wurde.

Vergleich der Datenbank

Im oben beschriebenen Verfahren vergleichen wir eine Datenbank mit einem bereitgestellten Schema. Sollten Unterschiede bestehen, führen wir einen weiteren Vergleich durch – mit dem Ziel, zusätzliche Spalten, Indexe und/oder Constraints zu entfernen.

Wie bereits beschrieben müssen wir beim Entfernen solcher zusätzlichen Elemente auf die Reihenfolge achten, in der die Elemente entfernt werden.

Dazu beschreiben wir ein weiteres Beispiel: stellen wir uns vor es existiert eine zusätzliche Spalte, die Teil eines primären Schlüssels ist, gleichzeitig existiert aber auch ein Fremdschlüssel, der diesen primären Schlüssel referenziert. Um eine solche Spalte entfernen zu können, muss also zunächst der Fremdschlüssel entfernt werden, dann der primäre Schlüssel und zu guter Letzte die Spalte selbst.

Daraus ergibt sich die folgende Reihenfolge für die Ausführung des Vergleichs: Constraints (erst Fremdschlüssel, dann der Rest), Indexe und letztendlich die Spalten. Bei jedem Vergleich wird dabei ein einziges Element entfernt (falls etwas entfernt werden muss).

Zur Durchführung benutzen wir die Funktion **CompareColumnsFromTable()**. Die Funktion besitzt einen Parameter, der bestimmt, ob zusätzliche Elemente entfernt werden sollen. Der Parameter trägt den Namen **remove**.

```
private void CompareColumnsFromTable(string tableName, JArray tableDb, JArray
tableJson,
            bool remove)
{
    JObject missingColumnsFromDb = new JObject();
    JObject missingColumnsFromJson = new JObject();
    JObject differentColumns = new JObject();

    // Compare each column of database with each column of schema
    foreach(var (coldb, colnamedb) in from JToken coldb in tableDb
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
                                  let colnamedb = (string)coldb["Column Name"]
                                  select (coldb, colnamedb))
{
    bool found = false;
    foreach(var coljs in from JToken coljs in tableJson
                         let colnamejs = (string)coljs["Column Name"]
                         where colnamejs != null && colnamedb == colnamejs
                         select coljs)
    {
        found = true;
        if(!JToken.DeepEquals(coldb, coljs)) // we have differences
        {
            _diffmsg += "\r\nIn table " + tableName + " column [" +
            colnamedb + "] is different in database";

            // Nqw we store the full columns definition
            // We need this when deciding if column change is possible
            JObject differences = new JObject
            {
                ["database"] = coldb,
                ["json"] = coljs
            };
                          _haveDiff = true;
              if(remove)
            differentColumns[colnamedb] = differences.DeepClone();
                    // See if the column (which will be changed) is part of a
                       constraint or index constraint
                            RemoveConstraintIfContainsColumn(tableName,
                            colnamedb);
                            RemoveIndexIfContainsColumn(tableName,
                            colnamedb);
                          }
            break;
        }
    }
    if(!found)
    {
        _diffmsg += "\r\nIn table " + tableName + " column [" +
            colnamedb + "] is missing from schema file";
        _haveDiff = true;
                          if(remove)
            missingColumnsFromJson[colnamedb] = coldb;
              // See if the column (which will be changed) is part of a
                 constraint or index contraint
                    RemoveConstraintIfContainsColumn(tableName,
                    colnamedb););););););););););););););
                    RemoveIndexIfContainsColumn(tableName, colnamedb);
            }
    }
}

// Compare each column of schema with each column of database
foreach(var (coljs, colnamejs) in from JToken coljs in tableJson
                                  let colnamejs = (string)coljs["Column
                                  Name"]
                                  select (coljs, colnamejs))
{
    bool found = false;

    // Here we look ONLY for columns in json file which are NOT in the
       database
    // These columns need to be added in the database
    foreach(var _ in from JToken coldb in tableDb
                     let colnamedb = (string)coldb["Column Name"]
                     where colnamejs == colnamedb
                     select new
                     {
                     })
    {
        found = true;
        break;
    }
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
if(!found)
{
    _diffmsg += "\r\nIn table " + tableName + " column [" +
        colnamejs + "] is missing from database";
            _haveDiff = true;
                  if(remove)
                  missingColumnsFromDb[colnamejs] = coljs;
}
}
                  if(remove)
                  {
                  // Add elements collected
                  if(missingColumnsFromDb.Count > 0)
                      _extraColumnsInJson[tableName] =
                      missingColumnsFromDb.DeepClone();
          if(missingColumnsFromJson.Count > 0)
                      _extraColumnsInDb[tableName] =
                      missingColumnsFromJson.DeepClone();
          if(differentColumns.Count > 0)
                      _differentColumns[tableName] = differentColumns.
                      DeepClone();
                                      }
}
```

Nachdem die zusätzlichen Tabellen entfernt wurden, können auch die fehlenden Tabellen erstellt werden. Der Ablauf dafür ist analog zu dem zuvor vorgestellten Vorgang des Erstellens der Tabellen beim Reproduzieren der Datenbank basierend auf einem Schema.

Anpassen der Spalten

Mit dem Erstellen von neuen Tabellen innerhalb der Datenbank muss sichergestellt sein, dass die existierenden Spalten identisch sind. Das bedeutet, fehlende Spalten müssen hinzugefügt, zusätzliche entfernt und Veränderungen angepasst werden.
Eine Besonderheit, die dabei auftreten kann, sind Spalten, die mit Hilfe von Always Encrypted verschlüsselt wurden. Dabei sind drei spezielle Fälle zu beachten:

- Die Spalten sind verschlüsselt und die Schlüssel sind nicht die selben → Ein Fehler wird entstehen
- Falls die Spalte nur im Schema verschlüsselt wird, wird sie in der Datenbank verschlüsselt sein
- Falls die Spalte nur in der Datenbank verschlüsselt wird, wir sie im Schema entschlüsselt sein

In unserem Verfahren werden wir nun die en-/decryption der Spalte betrachten. Die Idee ist dabei relativ simpel: wir erstellen eine temporären Tabelle, in der die Spalte entweder verschlüsselt oder entschlüsselt vorliegt (basierend auf der Vorgabe des Schemas), kopieren die Spalte von der Tabelle in diese temporäre Tabelle, entfernen die Spalte von der Tabelle und stellen die Spalte anschließend basierend auf den Informationen des Schemas wieder her. Zu guter Letzt können die Werte aus der temporären Tabelle wieder in die neu erstellte Spalte übertragen werden.

Mit folgendem Code kann das Vorgehen umgesetzt werden:

```
/*
 * We try to encrypt or decrypt a column. This is done as follow:
 *
 *  - create a temporary table with that column
 *  - copy the column into the new table
 *  - delete the existing column from the table
 *  - create the column according to the new rules
 *  - copy the column from temp table back
 *  - delete temp table
 *
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
 * We need to build the correspoding strings for both columns:
 *  - for the column in the table ADD COLUMN according to the definition in the
      json
 *  - for the column in the temp table with no encryption, regardless the status
      of the
 *    column in the table
*/
if(!string.IsNullOrWhiteSpace(encryptionTypeDb) || !string.IsNullOrWhiteSpace
(encryptionTypeJson))
{
    SqlTransaction transaction = _conn.BeginTransaction();
    cmd.Transaction = transaction;
    try
    {
        // Create a temp table and copy table into it
        string tmptablename = " [$__" + tableName.Replace("[", "").Replace("]",
        "") + "__$]";
        query = "Drop Table IF EXISTS " + tmptablename + ";\n";
        query += "select * into " + tmptablename + " From " + tableName;
        cmd.CommandText = query;
        cmd.ExecuteNonQuery();

        // Remove column from table
        query = "Alter table " + tableName + " drop column IF EXISTS [" +
        columnName + "];";
        cmd.CommandText = query;
        cmd.ExecuteNonQuery();

        // Empty original table
        query = "Truncate table " + tableName;
        cmd.CommandText = query;
        cmd.ExecuteNonQuery();

        // Add the column according to the definition in json file
        if(!string.IsNullOrWhiteSpace(encryptionTypeJson))
            encrypttext=" COLLATE Latin1_General_BIN2 ENCRYPTED WITH (COLUMN_
            ENCRYPTION_KEY=["+
                _cek + "], ENCRYPTION_TYPE = " + encryptionTypeJson +
                ", ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256')";
        else
            encrypttext = string.Empty;

        query = "ALTER TABLE " + tableName + " ADD [";
        query += columnName + "] " + datatypejson;

        if(precisionJson.HasValue)
        {
            query += "(" + precisionJson.Value;
            if(scaleJson.HasValue)
                query += ", " + scaleJson.Value;
            query += ") ";
        }
        else if(maxLengthJson.HasValue)
            query += "(" + maxLengthJson.Value + ") ";
        if(!isNullableJson)
            query += " NOT NULL";
        if(!string.IsNullOrWhiteSpace(encrypttext))
            query += encrypttext;
        if(!isNullableJson)
            query += " NOT NULL";

        cmd.CommandText = query;
        cmd.ExecuteNonQuery();

        // Copy back the column from temp table using bulk copy
        query = "select * from " + tmptablename + ";";
        cmd.CommandText = query;
        using DataTable dt = new DataTable();
        using(SqlDataAdapter adapter = new SqlDataAdapter(cmd))
        {
```


Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612

```
            cmd.CommandType = CommandType.Text;
            adapter.SelectCommand.CommandTimeout = 240;
            adapter.Fill(dt);
        }
        using SqlBulkCopy bulkCopy = new SqlBulkCopy(_conn,
            SqlBulkCopyOptions.AllowEncryptedValueModifications | SqlBulkCopy-
            Options.KeepIdentity,
            transaction);
        bulkCopy.DestinationTableName = tableName;
        bulkCopy.BatchSize = 1000;
        bulkCopy.BulkCopyTimeout = 240;
        bulkCopy.NotifyAfter = 1000;
        bulkCopy.WriteToServer(dt);

        // Remove the temporary table
        query = "Drop Table IF EXISTS " + tmptablename + ";";
        cmd.CommandText = query;
        cmd.ExecuteNonQuery();

        transaction.Commit();
    }
    catch(Exception e)
    {
        _error += "\r\nError changing column [" + columnName + "] in table " +
         tableName + ": " +
            e.Message;
        _haveErrors = true;
        transaction.Rollback();
    }
    cmd.Transaction = null;
    continue;
}
```

Automatisierte Lösungen

Das in diesem Artikel vorgestellte Verfahren stellt ein gewisses Maß an Vorwissen und Erfahrung in SQL Server und C# voraus. Und auch für erfahrene Benutzer ist sowohl der Aufbau als auch die Durchführung einer solchen Aufgabe keine triviale Sache. Immer häufiger wird in der Industrie aus diesem Grund auf Automatisierungen und Werkzeuge zurückgegriffen, um den Benutzer zu entlasten.

Eines dieser Werkzeuge ist unsere hauseigene Software SQLSyncer. Der SQLSyncer kann zur Migration und kontinuierlichen Integration von Datenbank-Objekten unterschiedlicher SQL Server verwendet werden. Bei der Durchführung des hier beschriebenen Verfahrens muss nur noch eine Auswahl der Datenbank-Objekte getroffen werden – der Rest passiert wie von selbst. Das Gute: mehr Zeit für die wichtigen Dinge!

Mehr Informationen zum SQLSyncer und Migration/Synchronisation von SQL Server Datenbanken findest Du auf unserer Website.

c#, csharp, datenbank, migration, sql server, synchronisation

Mainzer Datenfabrik GmbH
Rotterdamer Platz 2, 55120 Mainz
info@madafa.de
madafa.de
+49 6131 3331612