Save SQL Server Database Structure as Json

By: Mircea Dragan | Updated: 2020-01-13 | Comments | Related: More > Scripts

Problem

There are situations when we need the structure of a SQL Server database and database objects for further processing. For example, when we want to create a skeleton of the database in a different location (i.e. computer, SQL Server, etc.), or when we want to make sure that the database structure has not been altered.

Solution

There are several ways of obtaining database schema. Sometimes it is important how we arrange the information, because when we process it later on it will be easier compared to the case when we have the information in a "messy" format.

In this tip we present how to get SQL Server columns, primary keys, other constraints and indexes and present the data in a JSON format. Also, it is not difficult to obtain other database objects (like stored procedures, etc.) in a similar way.

Generate SQL Server Table Structure in JSON Format

A nice layout of tables and their columns looks like:

```
{
 "Table Schema": <schema name>,
 "Tables": [
  {
   "Table Name": <table name>,
   "Table Columns": [
    {
     "Column Name": <column name>,
     "is_nullable": false,
     "is_identity": true,
     "Column Type": [
      {
       "data_type": "int",
       "seed_value": 1,
       "increment_value": 1
      }
     ]
    }
   ]
  }
 ]
}
```

We group tables in the same schema, and columns in the same table.

The SQL SELECT statement which does this is:

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

As you can see, we collect only partial information about a column. It is not difficult to include other column properties if necessary. Also, we collect information about encrypted columns, so we assume the SQL Server version is at least 2017

Saving a generated JSON file is easy and there are several ways how to do it.

Generate SQL Server Foreign Keys Structure in JSON Format

A nice layout of foreign keys looks like:

```
{
 "Table Schema": <table schema>,
 "Table Name": <table name>,
 "Constraints": [
  {
   "Constraint Name": <FK name>,
   "Referenced Schema": <referenced schema name>,
   "Referenced Table": <referenced table name>,
   "Columns": [
    {
     "COLUMN_NAME": <column name>,
     "ordinal_position": 1,
     "Referenced Column": 2
    },
    {
     "COLUMN_NAME": <column name>,
     "ordinal_position": 2,
     "Referenced Column": 3
    }
   ]
  }
 ]
}
```

We group foreign keys in a similar way as for tables. Because a foreign key can refer to a key defined in another schema, we also need that information. Collecting the information in this format is more complex than for tables. The information we collect is the definition of the foreign key, such that we can easily replicate the foreign key if we want to create it in another database.

The SQL SELECT statement which does this is:

```
select distinct
   ConstraintColumns.TABLE_SCHEMA [Table Schema],
   ConstraintColumns.Table_Name [Table Name],
   [Constraints].CONSTRAINT_NAME [Constraint Name],
   [Constraints].[Referenced Schema],
   [Constraints].[Referenced Table],
   Columns.COLUMN_NAME,
   Columns.ordinal_position, Columns.[Referenced Column]
```

```
from INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE ConstraintColumns
join
    (
        select
            tc.TABLE_SCHEMA,
            tc.TABLE_NAME,
            tc.CONSTRAINT_NAME,
            rc.CONSTRAINT_SCHEMA [Referenced Schema],
            ccolumns.[Referenced Table]
        from INFORMATION_SCHEMA.TABLE_CONSTRAINTS tc
        join INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS rc
            on tc.CONSTRAINT_SCHEMA=rc.CONSTRAINT_SCHEMA
                and tc.CONSTRAINT_NAME=rc.CONSTRAINT_NAME
        join
            (
                select
                    schema_name(fk.schema_id) Table_Schema,
                    object_name(fkc.parent_object_id) Table_Name,
                    object_name(fkc.constraint_object_id) Constraint_Name,
                    object_name(fkc.referenced_object_id) [Referenced Table]
                from sys.foreign_key_columns fkc
                join sys.foreign_keys fk on fk.object_id=fkc.constraint_object_id
                join INFORMATION_SCHEMA.KEY_COLUMN_USAGE kcu
                    on kcu.TABLE_NAME=object_name(fk.parent_object_id)
                        and kcu.ORDINAL_POSITION=fkc.parent_column_id
                        and kcu.CONSTRAINT_NAME=object_name(fkc.constraint_object_id)
            ) ccolumns
                on tc.TABLE_SCHEMA=ccolumns.TABLE_SCHEMA
                    and tc.TABLE_NAME=ccolumns.TABLE_NAME
    ) [Constraints]
        on ConstraintColumns.CONSTRAINT_NAME=[Constraints].CONSTRAINT_NAME
            and ConstraintColumns.TABLE_NAME=[Constraints].TABLE_NAME
join
    (
        select
            ConstraintColumns.TABLE_SCHEMA,
            ConstraintColumns.TABLE_NAME,
            ConstraintColumns.COLUMN_NAME,
            ConstraintColumns.CONSTRAINT_NAME,
            ConstraintColumns.ordinal_position,
            ConstraintColumns.[Referenced Column]
        from
            (
                select
                    schema_name(fk.schema_id) Table_Schema,
                    object_name(fkc.parent_object_id) Table_Name,
                    object_name(fkc.constraint_object_id) Constraint_Name,
                    kcu.COLUMN_NAME,
                    fkc.constraint_column_id ORDINAL_POSITION,
                    fkc.parent_column_id [Referenced Column]
                from sys.foreign_key_columns fkc
                join sys.foreign_keys fk on fk.object_id=fkc.constraint_object_id
                join INFORMATION_SCHEMA.KEY_COLUMN_USAGE kcu
                    on kcu.TABLE_NAME=object_name(fk.parent_object_id)
```

```
                and kcu.ORDINAL_POSITION=fkc.parent_column_id
                and kcu.CONSTRAINT_NAME=object_name(fkc.constraint_object_id)
        ) ConstraintColumns
    ) Columns
      on ConstraintColumns.TABLE_SCHEMA=Columns.TABLE_SCHEMA
        and ConstraintColumns.TABLE_NAME=Columns.TABLE_NAME
order by ConstraintColumns.TABLE_SCHEMA, ConstraintColumns.Table_Name,
    [Constraint Name], [Referenced Schema], [Referenced Table],
    Columns.ordinal_position
for json auto, root('DBFKConstraints')
```

It was difficult to collect information about all constraints in one SELECT statement, but you can feel free to try.

Generate Other SQL Server Object Constraints Structure in JSON Format

Other constraints are primary keys, check constraints and unique keys. A nice layout of these constraints looks like:

```
{
 "Table Schema": <schema name>,
 "Table Name": <table name>,
 "Constraints": [
  {
   "Constraint Name": <constraint name>,
   "Constraint Type": "PRIMARY KEY",
   "Columns": [
    {
     "COLUMN_NAME": <column name>,
     "ordinal_position": 1
    }
   ]
  },
  {
   "Constraint Name": <constraint name>,
   "Constraint Type": "UNIQUE",
   "Columns": [
    {
     "COLUMN_NAME": <column name>,
     "ordinal_position": 1
    },
    {
     "COLUMN_NAME": <column name>,
     "ordinal_position": 2
    }
   ]
  }
 ]
},
{
 "Table Schema": <schema name>,
 "Table Name": <table name>,
 "Constraints": [
  {
   "Constraint Name": <constraint name>,
   "Constraint Type": "CHECK",
   "CHECK_CLAUSE": <check clause>,
   "Columns": [
    {
     "COLUMN_NAME": <column name>
    }
   ]
  }
 ]
}
```

These constraints are also grouped in a similar way as foreign keys.

The SQL SELECT statement which does this is:

```
select distinct
```

```
    ConstraintColumns.TABLE_SCHEMA [Table Schema],
    ConstraintColumns.Table_Name [Table Name],
    [Constraints].CONSTRAINT_NAME [Constraint Name],
    [Constraints].CONSTRAINT_TYPE [Constraint Type],
    [Constraints].CHECK_CLAUSE,
    Columns.COLUMN_NAME,
    Columns.ordinal_position
from INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE ConstraintColumns
join
    (
        select tc.TABLE_SCHEMA, tc.CONSTRAINT_CATALOG, tc.CONSTRAINT_NAME,
            tc.CONSTRAINT_TYPE, tc.TABLE_NAME, ck.CHECK_CLAUSE
        from INFORMATION_SCHEMA.TABLE_CONSTRAINTS tc
        left join INFORMATION_SCHEMA.CHECK_CONSTRAINTS ck
            on tc.CONSTRAINT_CATALOG=ck.CONSTRAINT_CATALOG
                and tc.CONSTRAINT_NAME=ck.CONSTRAINT_NAME
    ) [Constraints]
        on ConstraintColumns.CONSTRAINT_NAME=[Constraints].CONSTRAINT_NAME
            and ConstraintColumns.CONSTRAINT_CATALOG=[Constraints].CONSTRAINT_CATALOG
            and ConstraintColumns.TABLE_NAME=[Constraints].TABLE_NAME
join
    (
        select CheckColumns.TABLE_SCHEMA,
            CheckColumns.TABLE_NAME, CheckColumns.COLUMN_NAME,
            CheckColumns.CONSTRAINT_NAME, null as ordinal_position
        from INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE CheckColumns
        where CheckColumns.CONSTRAINT_NAME in
            (
                select CONSTRAINT_NAME from INFORMATION_SCHEMA.CHECK_CONSTRAINTS
            )
        union all
        select
            ConstraintColumns.TABLE_SCHEMA,
            ConstraintColumns.TABLE_NAME, ConstraintColumns.COLUMN_NAME,
            ConstraintColumns.CONSTRAINT_NAME, ConstraintColumns.ordinal_position
        from
            (
                select
                    kcu.TABLE_SCHEMA,
                    kcu.TABLE_NAME,
                    kcu.CONSTRAINT_NAME,
                    kcu.COLUMN_NAME,
                    kcu.ORDINAL_POSITION
                from
                    (
                        select * from INFORMATION_SCHEMA.KEY_COLUMN_USAGE
                        where CONSTRAINT_NAME not in
                            (
                                select object_name([constraint_object_id])
                                from sys.foreign_key_columns
                            )
                     ) kcu
        ) ConstraintColumns
    ) Columns
```

```
        on ConstraintColumns.TABLE_SCHEMA=Columns.TABLE_SCHEMA
            and ConstraintColumns.TABLE_NAME=Columns.TABLE_NAME
            and ConstraintColumns.CONSTRAINT_NAME=Columns.CONSTRAINT_NAME
order by
    ConstraintColumns.TABLE_SCHEMA,
    ConstraintColumns.Table_Name,
    [Constraint Name],
    Columns.ordinal_position
for json auto, root('DBConstraints')
```

As you can see, we collect not only the constraints names, but also their definitions, such that it is easy to replicate them in another database.

Generate SQL Server Index Structure in JSON Format

We use another query for indexes. A nice layout of indexes looks like:

```
{
 "Table Schema": <schema name>,
 "Table Name": <table name>,
 "Indexes": [
  {
   "index_name": <index name>,
   "index_type": "IX",
   "type_desc": "NONCLUSTERED",
   "ignore_dup_key": false,
   "idxcol": [
    {
     "column_id": 1,
     "column_name": <column name>,
     "is_descending_key": false
    }
   ]
  },
  {
   "index_name": <index name>,
   "index_type": "PK",
   "type_desc": "CLUSTERED",
   "ignore_dup_key": false,
   "idxcol": [
    {
     "column_id": 1,
     "column_name": <column name>,
     "is_descending_key": false
    }
   ]
  },
  {
   "index_name": <index name>,
   "index_type": "UQ",
   "type_desc": "NONCLUSTERED",
   "ignore_dup_key": false,
   "idxcol": [
    {
     "column_id": 1,
     "column_name": <column name>,
     "is_descending_key": false
    },
    {
     "column_id": 2,
     "column_name": <column name>,
     "is_descending_key": false
    }
   ]
  }
 ]
}
```

Indexes are also grouped in a similar way as the rest of elements presented above.

In SSMS the indexes shown for a particular table contain proper indexes (created with command CREATE INDEX), as well as primary keys and unique constraints. Our SELECT statement does the same.

The SQL SELECT statement is:

```
select
   schema_name(obj.schema_id) [Table Schema],
   obj.name [Table Name],
   [Indexes].name index_name,
   (
      case when is_primary_key=1
         then 'PK'
         else
            case when is_unique_constraint=1
               then 'UQ'
               else 'IX'
            end
      end
   ) index_type,
   [Indexes].type_desc,
   index_column_id [column_id],
   (
      select name from sys.columns cols
      where cols.object_id=obj.object_id and idxcol.column_id=cols.column_id
   ) column_name,
   ignore_dup_key,
   is_descending_key
from sys.indexes [Indexes]
   join sys.objects obj on obj.object_id=[Indexes].object_id and obj.type='u'
   join sys.index_columns idxcol on obj.object_id=idxcol.object_id
      and idxcol.index_id=[Indexes].index_id
where is_disabled=0
order by [Table Schema], [Table Name], index_name, column_id
for json auto, root('DBIndexes')
```

As you can see, we collect the definition for each index, such that it is easy to replicate them in another database.

Next Steps

- The queries above can be run on any SQL Server 2017 or newer. They were tested (and currently used) on SQL Server 2017 and 2019.
- The JSON file(s) can be created in several ways, one way is using SSMS: run the query against database of your choice, click on the link generated by the query and copy/paste the content opened in a new window to a Json file. You will need to format the result in order to get the format presented above. In another tip we will present a way of saving the schema using a C# program.
- These queries will be used in comparing two database structures and creating a new database based on the schema provided in the JSON file.

Last Updated: 2020-01-13

About the author

Mircea Dragan is a Senior Computer Scientist with a strong mathematical background with over 18 years of hands-on experience.

View all my tips

Related Resources

More Database Developer Tips...