How to Generate a CREATE T Existing Table: Part 1

Sergey Syrovatchenko

Last updated date Jun 05 2019

SQL Server stores information about all objects and their properties as metadata that can be accessed through system views. In addition, some of the system views hold interesting nuances that can help to better understand how a DBMS works.

To see the system view body, just as for any other script object, the *OBJECT_DEFINITION* function is used*:*

```
PRINT OBJECT_DEFINITION(OBJECT_ID('sys.objects'))
```

However, *OBJECT_DEFINITION*, as well as its analogue *sp_helptext*, has a significant disadvantage; it does not allow the return script description for a table object.

```
IF OBJECT_ID('dbo.Table1', 'U') IS NOT NULL
  DROP TABLE dbo.Table1
GO

CREATE TABLE dbo.Table1 (ColumnID INT PRIMARY KEY)
GO

EXEC sys.sp_helptext 'dbo.Table1'
SELECT OBJECT_DEFINITION(OBJECT_ID('dbo.Table1', 'U'))
```

When executing *sp_helptext*, we will get the following error:

Msg 15197, Level 16, State 1, Procedure sp_helptext, Line 107
There is no text for object 'dbo.Table1'.

Under the same conditions, a system function *OBJECT_DEFINITION* returns *NULL*.

Fetching from sys.sql_modules will also not solve the problem, since the same old OBJECT_DEFINITION function call is used inside this system view:

```
CREATE VIEW sys.sql_modules AS
```

```
    SELECT object_id = o.id,
        definition = object_definition(o.id),
```

Such behavior is rather disappointing. Sometimes it is nee ry to retrieve a scrip description of a table for some scripts. Well, let's look at s n *OBJECT_DEFINITION* function analogue for working with table objects.

```
IF OBJECT_ID('dbo.WorkOut', 'U') IS NOT NULL
    DROP TABLE dbo.WorkOut
GO

CREATE TABLE dbo.WorkOut
(
    WorkOutID BIGINT IDENTITY(1,1) NOT NULL,
    TimeSheetDate AS DATEADD(DAY, -(DAY(DateOut) - 1), DateOut),
    DateOut DATETIME NOT NULL,
    EmployeeID INT NOT NULL,
    IsMainWorkPlace BIT NOT NULL DEFAULT 1,
    DepartmentUID UNIQUEIDENTIFIER NOT NULL,
    WorkShiftCD NVARCHAR(10) NULL,
    WorkHours REAL NULL,
    AbsenceCode VARCHAR(25) NULL,
    PaymentType CHAR(2) NULL,
    CONSTRAINT PK_WorkOut PRIMARY KEY CLUSTERED (WorkOutID)
)
GO
```

And proceed to the first step, getting a list of columns and their properties.

Essentially, the list of columns can be obtained by simply referencing one of the several system views. Thus, it is important to fetch from the simplest system views, in order for the query execution time to be minimal.

Here are a few examples along with their execution plans made in dbForge Studio for SQL Server:

```
--#1
SELECT *
FROM INFORMATION_SCHEMA.COLUMNS c
WHERE c.TABLE_SCHEMA = 'dbo'
    AND c.TABLE_NAME = 'WorkOut'
```

```sql
--#2
SELECT c.*
FROM sys.columns c WITH(NOLOCK)
JOIN sys.tables t WITH(NOLOCK) ON c.[object_id] = t.[object_id]
JOIN sys.schemas s WITH(NOLOCK) ON t.[schema_id] = s.[schema_id]
WHERE t.name = 'WorkOut'
    AND s.name = 'dbo'
```

```sql
--#3
SELECT *
FROM sys.columns c WITH(NOLOCK)
WHERE OBJECT_NAME(c.[object_id]) = 'WorkOut'
    AND OBJECT_SCHEMA_NAME(c.[object_id]) = 'dbo'
```

```sql
--#4
SELECT *
FROM sys.columns c WITH(NOLOCK)
WHERE c.[object_id] = OBJECT_ID('dbo.WorkOut', 'U')
```

The presented plans show that the #1 and #2 approaches contain excessive amount of connections that will increase the query execution time, while the #3 approach leads to the complete scan of the index, making it the least efficient of all.

In terms of performance, the #4 approach remains the most attractive to me.

However, data contained in *sys.columns* (as well as in *INFORMATION_SCHEMA.COLUMNS*) is not enough to completely describe the table structure. This forces joins to other system

```
SELECT
      c.name
    , [type_name] = tp.name
    , type_schema_name = s.name
      c.max_length
    ,
    , c.collation_name
    , c.is_nullable
    , c.is_identity
    , ic.seed_value
    , ic.increment_value
    , computed_definition = cc.[definition]
    , default_definition = dc.[definition]
FROM sys.columns c WITH(NOLOCK)
JOIN sys.types tp WITH(NOLOCK) ON c.user_type_id = tp.user_type_id
JOIN sys.schemas s WITH(NOLOCK) ON tp.[schema_id] = s.[schema_id]
LEFT JOIN sys.computed_columns cc WITH(NOLOCK) ON
        c.[object_id] = cc.[object_id]
    AND c.column_id = cc.column_id
LEFT JOIN sys.identity_columns ic WITH(NOLOCK) ON
        c.[object_id] = ic.[object_id]
    AND c.column_id = ic.column_id
LEFT JOIN sys.default_constraints dc WITH(NOLOCK) ON dc.[object_id] = c.default_object_id
WHERE c.[object_id] = OBJECT_ID('dbo.WorkOut', 'U')
```

Accordingly, the execution plan will look not so optimistic, as before. Note that the column list is even read out 3 times:

Have a look inside *sys.default_constraints*:

```
ALTER VIEW sys.default_constraints AS
    SELECT name, object_id, parent_object_id,
        ...
        object_definition(object_id) AS definition,
        is_system_named
    FROM sys.objects$
    WHERE type = 'D ' AND parent_object_id > 0
```

There is an *OBJECT_DEFINITION* call inside the system view. So, to retrieve the description of the default constraint, we don't need to establish joining.

OBJECT_DEFINITION is still used in *sys.computed_columns*:

```
ALTER VIEW sys.computed_columns AS
    SELECT object_id = id,
        name = name,
        column_id = colid,
        system_type_id = xtype,
        user_type_id = utype,
```

```
    ...
  FROM sys.syscolpars
  WHERE number = 0
    AND (status & 16) = 16 -- CPM_COMPUTED
    AND has_access('CO', id) = 1
```

We seem to have already avoided 2 joins. The case with *sys.identity_columns* is more curious:

```
ALTER VIEW sys.identity_columns AS
  SELECT object_id = id,
    name = name,
    column_id = colid,
    system_type_id = xtype,
    user_type_id = utype,
    ...
    seed_value = IdentityProperty(id, 'SeedValue'),
    increment_value = IdentityProperty(id, 'IncrementValue'),
    last_value = IdentityProperty(id, 'LastValue'),
    ...
  FROM sys.syscolpars
  WHERE number = 0 -- SOC_COLUMN
    AND (status & 4) = 4 -- CPM_IDENTCOL
    AND has_access('CO', id) = 1
```

To retrieve information about *IDENTITY* properties, an undocumented property *IDENTITYPROPERTY* is used. After a check, its unchanging behavior on SQL Server 2005 and higher was ascertained.

As a result of calling these functions directly, the column list obtaining query becomes significantly simplified:

```
SELECT
   c.name
  , [type_name] = tp.name
  , type_schema_name = s.name
  , c.max_length
  , c.[precision]
  , c.scale
  , c.collation_name
  , c.is_nullable
  , c.is_identity
  , seed_value = CASE WHEN c.is_identity = 1 THEN IDENTITYPROPERTY(c.
[object_id], 'SeedValue') END
  , increment_value = CASE WHEN c.is_identity = 1 THEN IDENTITYPROPER
[object_id], 'IncrementValue') END
  , computed_definition = OBJECT_DEFINITION(c.
[object_id], c.column_id)
  , default_definition = OBJECT_DEFINITION(c.default_object_id)
FROM sys.columns c WITH(NOLOCK)
```

```
JOIN sys.types tp WITH(NOLOCK) ON c.user_type_id = tp.user_type_id
JOIN sys.schemas s WITH(NOLOCK) ON tp.[schema_id] = s.[schema_id]
WHERE c.[object_id] = OBJECT_ID('dbo.WorkOut', 'U')
```

And the execution plan becomes more efficient:

that triggers much more faster than JOIN. This is true, provided that the number of schemes does not exceed the number of user objects. And since such a situation is unlikely, it can be neglected.

Next, get a list of columns included in the primary key. The most obvious approach is to use *sys.key_constraints*:

```
SELECT
      pk_name = kc.name
    , column_name = c.name
    , ic.is_descending_key
FROM sys.key_constraints kc WITH(NOLOCK)
JOIN sys.index_columns ic WITH(NOLOCK) ON
        kc.parent_object_id = ic.object_id
    AND ic.index_id = kc.unique_index_id
JOIN sys.columns c WITH(NOLOCK) ON
        ic.[object_id] = c.[object_id]
    AND ic.column_id = c.column_id
WHERE kc.parent_object_id = OBJECT_ID('dbo.WorkOut', 'U')
    AND kc.[type] = 'PK'
```

In most cases, *PRIMARY KEY* is a clustered index and the *Unique* constraint.

At the metadata level, SQL Server sets index_id to 1 for all clustered indexes, so we can make a selection from sys.indexes filtering by is_primary_key = 1 or by index_id = 1 (not recommended).

Additionally, to avoid joining to *sys.columns*, the *COL_NAME* system function can be used:

```
SELECT
      pk_name = i.name
    , column_name = COL_NAME(ic.[object_id], ic.column_id)
    , ic.is_descending_key
FROM sys.indexes i WITH(NOLOCK)
JOIN sys.index_columns ic WITH(NOLOCK) ON
        i.[object_id] = ic.[object_id]
    AND i.index_id = ic.index_id
WHERE i.is_primary_key = 1
```

```
    AND i.[object_id] = object_id('dbo.WorkOut', 'U')
```

Now combine the obtained queries into one query to get t ollowing final query:

```
DECLARE
      @object_name SYSNAME
```

```

SELECT
      @object_name = '[' + OBJECT_SCHEMA_NAME(o.[object_id]) + '].[' + OBJECT_NAME([object_id]) + ']'
    , @object_id = [object_id]
FROM (SELECT [object_id] = OBJECT_ID('dbo.WorkOut', 'U')) o

SELECT @SQL = 'CREATE TABLE ' + @object_name + CHAR(13) + '(' + CHAR(13
    SELECT CHAR(13) + '    , [' + c.name + '] ' +
        CASE WHEN c.is_computed = 1
            THEN 'AS ' + OBJECT_DEFINITION(c.[object_id], c.column_id)
            ELSE
                CASE WHEN c.system_type_id != c.user_type_id
                    THEN '[' + SCHEMA_NAME(tp.[schema_id]) + '].[' + tp.name + ']'
                    ELSE '[' + UPPER(tp.name) + ']'
                END  +
                CASE
                    WHEN tp.name IN ('varchar', 'char', 'varbinary', 'b
                        THEN '(' + CASE WHEN c.max_length = -1
                                        THEN 'MAX'
                                        ELSE CAST(c.max_length AS VARCH
                                    END + ')'
                    WHEN tp.name IN ('nvarchar', 'nchar')
                        THEN '(' + CASE WHEN c.max_length = -1
                                        THEN 'MAX'
                                        ELSE CAST(c.max_length / 2 AS V
                                    END + ')'
                    WHEN tp.name IN ('datetime2', 'time2', 'datetimeoff
                        THEN '(' + CAST(c.scale AS VARCHAR(5)) + ')'
                    WHEN tp.name = 'decimal'
                        THEN '(' + CAST(c.[precision] AS VARCHAR(5)) + ',' + CAST(c.scale AS VARCHAR(5)) + ')'
                    ELSE ''
                END +
                CASE WHEN c.collation_name IS NOT NULL AND c.system_typ
                    THEN ' COLLATE ' + c.collation_name
                    ELSE ''
                END +
                CASE WHEN c.is_nullable = 1
                    THEN ' NULL'
                    ELSE ' NOT NULL'
                END +
                CASE WHEN c.default_object_id != 0
                    THEN ' CONSTRAINT [' + OBJECT_NAME(c.default_ob
                         ' DEFAULT ' + OBJECT_DEFINITION(c.default_obje
                    ELSE ''
                END +
```

```
[definition]
                    ELSE ''
                END +
                CASE WHEN c.is_identity = 1
                    THEN ' IDENTITY(' + CAST(IDENTITYPROPERTY(c
[object_id], 'IncrementValue') AS VARCHAR(5)) + ')'
                    ELSE ''
                END
        END
    FROM sys.columns c WITH(NOLOCK)
    JOIN sys.types tp WITH(NOLOCK) ON c.user_type_id = tp.user_type_id
    LEFT JOIN sys.check_constraints cc WITH(NOLOCK)
         ON c.[object_id] = cc.parent_object_id
        AND cc.parent_column_id = c.column_id
    WHERE c.[object_id] = @object_id
    ORDER BY c.column_id
    FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)'), 1, 7, '       '
    ISNULL((SELECT '
    , CONSTRAINT [' + i.name + '] PRIMARY KEY ' +
    CASE WHEN i.index_id = 1
        THEN 'CLUSTERED'
        ELSE 'NONCLUSTERED'
    END +' (' + (
    SELECT STUFF(CAST((
        SELECT ', [' + COL_NAME(ic.
[object_id], ic.column_id) + ']' +
                CASE WHEN ic.is_descending_key = 1
                    THEN ' DESC'
                    ELSE ''
                END
        FROM sys.index_columns ic WITH(NOLOCK)
        WHERE i.[object_id] = ic.[object_id]
            AND i.index_id = ic.index_id
        FOR XML PATH(N''), TYPE) AS NVARCHAR(MAX)), 1, 2, '')) + ')'
    FROM sys.indexes i WITH(NOLOCK)
    WHERE i.[object_id] = @object_id
        AND i.is_primary_key = 1), '') + CHAR(13) + ');'

PRINT @SQL
```

Which, when executed, will generate the following script for the test table:

```
CREATE TABLE [dbo].[WorkOut]
(
      [WorkOutID] [BIGINT] NOT NULL IDENTITY(1,1)
    , [TimeSheetDate] AS (dateadd(day, -(datepart(day,[DateOut])-(1)),
[DateOut]))
    , [DateOut] [DATETIME] NOT NULL
    , [EmployeeID] [INT] NOT NULL
    , [IsMainWorkPlace] [BIT] NOT NULL CONSTRAINT [DF__WorkOut__IsMainW
    , [DepartmentUID] [UNIQUEIDENTIFIER] NOT NULL
    , [WorkShiftCD] [NVARCHAR]
    , [AbsenceCode] [VARCHAR](25) COLLATE Cyrillic_General_CI_AS NULL
    , [PaymentType] [CHAR](2) COLLATE Cyri                          LL
    , CONSTRAINT [PK_WorkOut] PRIMARY KEY CLUSTERED ([WorkOutID])
);
```

As you can see, the topic is too broad and it is not limited to a column list and primary key.

That's why generation of indexes, foreign keys, and other statements are planned to be revealed in the next part of this topic.

> Next Recommended Article
>
> Creating Duplicate Table With New Name From Existing Table in SQL Server 2012
>
> In this article, you will see how to create a duplicate table with a new name using a script in SQL Server.

create table script | dynamic sql | metadata | sql | sql server | t-sql

Sergey Syrovatchenko

https://www.c-sharpcorner.com/members/sergey-syrovatchenko

1788 **205.1k**

17 **7**

Type your comment here and press Enter Key (Minimum 10 characters)

GREAT GREAT GREAT GENIUS

Karthik Karan Jul 30, 2019

1884 **4** **0** 0 0 Reply

Hi, the script works perfectly! however, in SQL Server 2016, there are tables that are memory optimized. which is not included in the result. can you add that also?

Jose Bohol Jun 20, 2018

1886 **2** **0** 0 0

Thanks

Nigel Fernandes
470 4k 562.7k
Sep 28, 2016
0 0 Reply

Good one
Santhakumar Munuswamy
35 33.9k 1m
Nov 14, 2015
0 Reply

Great Info! Is part 2 coming soon?
Todd Forsberg
1887 1 0
Feb 27, 2014
0 0 Reply

Nice Explaination
Anubhav Chaudhary
20 42k 14.4m
Dec 24, 2013
0 0 Reply

Welcome to C# Corner Sergey. Very nice and detailed explanation.
Mahesh Chand
Admin 379.7k 159.9m
Dec 23, 2013
0 0 Reply

TRENDING UP

01 Sealed Class Explained In C#

02 Learn Angular 8 Step By Step In 10 Days - HttpClient Or Ajax Call - Day Nine

03 How To Upload A File To Amazon S3 Using AWS SDK In MVC

04 Getting Started With .NET Core 3.1 - Part One

05 How To Manage Our Blob Storage Account Using Logic Apps

06 How To Create SSIS Catalog

07 What Can Be Done To Make Code Quality Better

08 A File System Manager From Scratch In .NET Core And VueJS

09 Learn About Extension Methods In C#

10 Localization In Blazor Server With .NET Core 3.1

View All

About Us Contact Us Privacy Policy Terms Media Kit Sitemap Report a Bug FAQ Partners
C# Tutorials Common Interview Questions Stories Consultants Ideas Certifications