**A Seven-Step Process for Changing a Database's Collation**

**Follow this simple methodology to prevent problems**

*Feb 24, 2014* **Stéphane Haby**

**FROM:** [**http://sqlmag.com/database-performance-tuning/seven-step-process-changing-database-collation**](http://sqlmag.com/database-performance-tuning/seven-step-process-changing-database-collation)

**NOTE THE METHOD IN THE COMMENT AT THE END OF THE ARTICLE WHICH IS SIMPLER PER SE**

Advertisement

With SQL Server, you can easily change the collation of a database. Before you do so, though, you should be familiar with the purpose and structure of collations, know what will be altered when the collation changes, and understand the consequences if a collation change isn't done correctly. It's also a good idea to have a methodology for changing the collation of a SQL Server database. I use a seven-step process, which I'll share with you.

**Related:** [Forcing Collation in the WHERE Clause](http://sqlmag.com/blog/forcing-collation-where-clause-22-jun-2011" \t "_blank)

**The Purpose and Structure of Collations**

Collation serves two important roles:

* It defines the rules that SQL Server applies when sorting and comparing characters in a database.
* It defines the underlying code pages used to physically store non-Unicode data (e.g., column types of char, varchar, and text).

Let's take a close look at each of these roles.

***Defining the sort and comparison rules.*** The rules that SQL Server applies when sorting and comparing characters are strongly related to the language associated with the data. They can also specify other sorting or comparison behaviors, such as whether the sort or comparison operation is case-sensitive (\_CS), case-insensitive (\_CI), accent-sensitive (\_AS), accent-insensitive (\_AI), or width-sensitive (\_WS). In SQL Server 2000 and later, you can use two types of collations:

* SQL Server collations, which start with the prefix SQL\_. These collations were used in SQL Server versions prior to SQL Server 2000. With these collations, the rules for Unicode and non-Unicode data are different; therefore, sorting and comparison operations on Unicode data will have different results than on non-Unicode data.
* Windows collations, which are the recommended collations. In particular, SQL Server 2008 introduced many Windows collations to support the Windows system locales. Because these collations are centralized at the OS level, they're common to all the applications installed on a Windows computer. As a result, they can provide useful and consistent sorting and comparison behaviors. In addition, the rules for Unicode and non-Unicode data are the same, ensuring data consistency.

***Defining the underlying code pages.*** A code page determines the way values are assigned to characters when data is stored on a disk or network. SQL Server supports 16 different code pages. They use single bytes to store each character. This limits the space and memory used and leads to better performance compared with using multi-bytes.

The drawback of the single-byte approach is the small number of available values: 28 or 256. As TechNet's [Code Page Architecture](http://technet.microsoft.com/en-us/library/ms186356(v=SQL.105).aspx" \t "_blank) web page explains:

There are 8 bits in a byte, and the 8 bits can be turned ON and OFF in 256 different patterns. A program that uses 1 byte to store each character can therefore represent up to 256 different characters by assigning a character to each of the bit patterns.

The values from b0000000 to b01111111 are used for all code pages equally. Their corresponding characters are encoded in a manner described by the ASCII base table. The other 128 bytes (represented by values b10000000 through b11111111) are individually mapped for each code page. The mapping between a value and a represented character can vary widely between code pages, so changing the collation might change the underlying code page.

If a code page change occurs, any stored data that's not defined in the ASCII base table will be converted. In some cases, characters that aren't defined in the new code page can't be converted and will be automatically changed to an "undefined" character. This means that some characters can get lost during a conversion. For this reason, if a code page change is needed, you need to analyze your data to avoid wrong conversions.

**Related:** [7 Considerations for Server Consolidations](http://sqlmag.com/database-administration/7-considerations-server-consolidations" \t "_blank)

Let's take a look at an example of an incorrect conversion that results in a character being lost. Suppose that the collation Latin1\_General\_CI\_AS is current. All the € characters stored in the char, varchar, and text columns are represented as the ASCII value 128. For some reason, the collation has to be changed to SQL\_Latin1\_General\_CP850\_CI\_AS. An analysis shows that this character isn't defined in the new collation. If the change is performed without any checks and corrections, all the € characters will be converted to the undefined character *?* with the ASCII character code of 63. As a result, users won't be able to tell the difference between the old € character and the real *?* character anymore. All of this information is lost. A possible solution to this problem is to change all of the € characters to *Euro* prior to the collation change.

To help you find the code pages used by different collations, you can use the [T-SQL](http://sqlmag.com/development/database-development/t-sql" \t "_blank) command

SELECT COLLATIONPROPERTY('<collation>', 'CodePage')

where *<collation>* is the name of the collation. It will generate a number referencing the code page associated with the specified collation. For example, the commands

SELECT COLLATIONPROPERTY('Traditional\_Spanish\_CS\_AS\_KS\_WS',  
  'CodePage')  
SELECT COLLATIONPROPERTY('Thai\_CS\_AS\_KS\_WS', 'CodePage')

return 1252 and 874, respectively.

Note that Unicode data stored in the column types of nchar, nvarchar, and ntext uses a UCS-2 code page (Universal Character Set, using only 2 bytes per character). This is identical for all collations in SQL Server 7.0 and later. This data doesn't need to be converted when the collation is changed because all Unicode data uses the same Unicode code page. Collations don't control the code page used for Unicode columns; they only control the attributes.

**Collation Recommendations**

When working with collations, there are a few recommendations to keep in mind:

* Use Windows collations to avoid having to use different rules for Unicode and non-Unicode data.
* Choose a collation that's a good match for not only the data to be stored but also the sorting and comparison operations you want to perform on that data.
* Don't mix collations at the instance level, database level, and object level (e.g., tables, store procedures), because doing so can strongly degrade the overall performance of the instance.
* Use a more general collation (e.g., latin1) when a database must contain data that's in different languages.
* Make sure that all characters to be stored in the database are defined in the underlining code page.

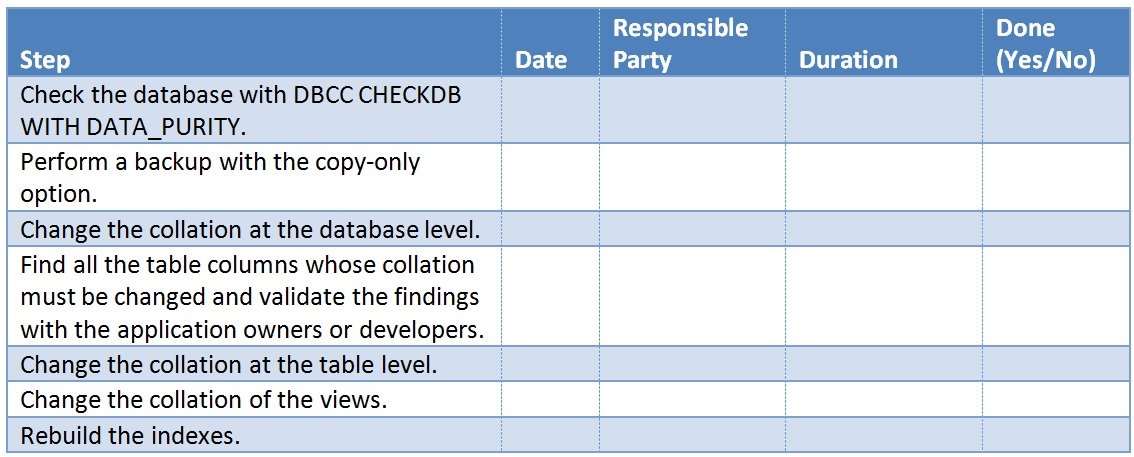
**Collation Change Methodology**

Before changing a database collation on a production system, you should test the change (including testing the applications that use the database) on a development, test, or quality assurance (QA) system. That way, you can avoid database problems and any surprises in applications' behavior. In addition, you should let the owners or developers of the applications know about the collation change beforehand.

I follow a seven-step process when changing a database collation:

1. Check the database with DBCC CHECKDB WITH DATA\_PURITY.
2. Perform a backup with the copy-only option.
3. Change the collation at the database level.
4. Find all the table columns whose collation must be changed and validate the findings with the owners or developers of the applications using the database.
5. Change the collation at the table level.
6. Change the collation of the views.
7. Rebuild the indexes.

Figure 1 shows the checklist I use to make sure I complete all the steps.



**Figure 1: Collation Change Checklist**

Let's look at each step in more detail.

**Step 1: Check the Database**

Although performing a complete integrity check is a good idea, it's not required. In many cases, simply using the DATA\_PURITY option can be enough:

DBCC CHECKDB WITH DATA\_PURITY

The DATA\_PURITY option will check the database for column values that are invalid or out-of-range (i.e., not defined in the code page). For example, it will tell you if value 128 is found, but the current collation doesn't define it.

**Step 2: Perform a Backup**

The next step is to make a backup of your database. You can perform a copy-only backup so that it won't affect your backup strategy. Here's a sample script you can use to perform the copy-only backup:

BACKUP DATABASE [DBxxx]  
  TO DISK = N'B:\Backup\DBxxx\_BeforeCollationChange.bak'  
  WITH COPY\_ONLY, NOFORMAT, NOINIT,  
  NAME = DBxxx\_BeforeCollationChange',  
  SKIP, NOREWIND, NOUNLOAD, STATS = 10

Before you run this script, you need to change each instance of *DBxxx* to the name of your database.

**Step 3: Change the Collation at the Database Level**

After you've backed up your database, you can change the database collation. To do so, use the command

ALTER DATABASE [DBxxx] COLLATE New\_Collation

where *DBxxx* is the name of your database and *New\_Collation* is the name of the collation you want to use.

In many cases, this command will run successfully. However, the command might fail and generate error messages concerning functions, primary keys, constraints, or indexes. These objects don't contain primary data, so they can be re-created.

If errors do occur, you need to:

* Drop all functions, primary keys, constraints, and indexes.
* Change the database collation.
* Re-create all the dropped objects. (Don't forget to reassign the rights granted to these objects.)

However, I recommend that you re-create all the dropped objects later in step 7. That way, you might avoid having to redo some work.

If this step goes well, the database properties will show the new collation. Many DBAs will stop here, but as you'll see shortly, this step only changes the default collation for the new objects that will be created in the database. All existing objects still contain data values referring to their origin mapping.

**Step 4: Find All the Table Columns Whose Collation Must Be Changed**

The collation of all table columns of types char, varchar, text, nchar, nvarchar, and ntext must be changed. To list all the table columns with their current collations, you can run the query:

SELECT t.name, c.name, c.collation\_name  
FROM sys.columns c INNER JOIN sys.tables t  
ON t.object\_id = c.object\_id  
WHERE c.object\_id  
  IN (SELECT object\_id FROM sys.objects WHERE type = 'U')  
  AND c.collation\_name != 'NULL'  
ORDER BY t.name,c.name

In the generated list, you need to look for these three cases:

* Case 1: Columns defined with the old default collation
* Case 2: Columns defined with a collation that differs from the old default collation
* Case 3: Columns defined with a user data type (UDT)

Before making any changes to these columns' collations, you should check with the application owners or developers to see whether the changes will generate bad behaviors in the applications. In addition, if a column falls under case 2, you should ask them why its collation differs from the old default collation. This information will help you determine if the column can be changed to the new default collation.

**Step 5: Change the Collation at the Table Level**

After the application owners or developers have given you the okay to change the collation of the identified columns, you can make the changes. How you make those changes depends on the three cases identified in step 4.

***Case 1.*** When a column is defined with the old collation, you can change the old collation to the new collation with the command:

ALTER TABLE [<Table>] ALTER COLUMN [<Column>] <Column\_Type>  
  COLLATE <New\_Collation>

Before you run this command, be sure to replace *<Table>, <Column>, <Column\_Type>,* and *<New\_Collation>* with your information.

If the column has constraints or is used by indexes, this command will generate some errors. The referencing constraints and indexes must be dropped, then re-created after the collation is changed.

***Case 2.*** When a column has been defined with a collation that differs from the old default collation, your plan of action depends on what you found out from the application owners or developers. Their explanation will tell you if the column's collation can or can't be changed to the new default collation.

If an application owner or developer doesn't know why a column's collation differs from the old default collation, you shouldn't make any changes if the application wasn't experiencing any bad behaviors or poor performance or if the problems were minimal. If the application was behaving or performing very poorly, you might consider changing the column's collation to the new default one and thoroughly testing the application to see if the situation improves.

***Case 3.*** Administrators sometimes use UDTs, which are based on the system data types (e.g., char, varchar), when several tables must store the same type of data in a column. They must then ensure that these columns have exactly the same data type, length, and nullability. Often, the columns must also have the same format (which is achieved through rules).

For example, suppose that a description column always has a size of 1,000 in the database. To make sure that all the description columns in all the tables have the same size, you can create a UDT named DescriptionString:

CREATE TYPE [dbo].[DescriptionString]  
  FROM [varchar](1000) NULL

Then when you're creating the tables, you can use this data type. For example, the following code creates a table named Country whose Description column uses the DescriptionString UDT:

CREATE TABLE [dbo].[Country]  
(  
  [isocode] [char] (2) NOT NULL,  
  [fullname] [dbo].[NameString] NULL,  
  [description][dbo].[DescriptionString]  
)  
ON [PRIMARY]

When a column is defined with a UDT, you can't use the ALTER TABLE command shown in case 1 to change the collation. Doing so will generate an error like this:

Msg 2715, Level 16, State 6, Line 1  
Column, parameter, or variable #12:  
Cannot find data type DescriptionString.

You need to use a workaround to change the collation of a table that contains a column whose data type is a UDT. The workaround is best explained with an example. Suppose you need to change the collation of the Country table, whose Description column uses the DescriptionString UDT. First, you need to create another table, Tmp\_Country, using a standard data type for the Description column:

CREATE TABLE [dbo].[Tmp\_Country]  
(  
  [isocode] [char] (2) NOT NULL,  
  [fullname] [varchar] (100) NULL,  
  [description] [varchar] (1000) NULL  
)  
ON PRIMARY

Next, you need to copy all the rows from the Country table to the Tmp\_Country table, then delete the Country table:

IF EXISTS(SELECT \* FROM dbo.Country)  
  EXEC('INSERT INTO  
    dbo.Tmp\_Country(isocode,fullname,description)  
    SELECT isocode,fullname,description  
    FROM dbo.Country WITH(HOLDLOCK TABLOCKX)')  
  
DROP TABLE dbo.Country

At this point, you can change the collation in the Tmp\_Country table using the ALTER TABLE command:

ALTER TABLE dbo.Tmp\_Country  
  ALTER COLUMN [fullname] varchar(100)  
  COLLATE Latin1\_General\_CS\_AS NULL  
ALTER TABLE dbo.Tmp\_Country  
  ALTER COLUMN [description] varchar(1000)  
  COLLATE Latin1\_General\_CS\_AS NULL

Once that's done, you can re-create the Country table using the DescriptionString UDT for the Description column:

CREATE TABLE [dbo].[Country]  
(  
  [isocode] [char] (2) NOT NULL,  
  [fullname] [dbo].[NameString] NULL,  
  [description][dbo].[DescriptionString]  
)  
ON [PRIMARY]

Finally, you can copy all rows from the Tmp\_Country table to the Country table, then delete the Tmp\_Country table:

IF EXISTS(SELECT \* FROM dbo.Tmp\_Country)  
  EXEC('INSERT INTO  
    dbo.Country(isocode,fullname,description)  
    SELECT isocode,fullname,description  
    FROM dbo.Country WITH(HOLDLOCK TABLOCKX)')  
  
DROP TABLE dbo.Tmp\_Country

Fortunately, you can use SQL Server Management Studio (SSMS) to change a column's data type, making this workaround much easier to accomplish. SSMS will automatically generate the scripts to change it. If you analyze the scripts, you'll see that a temporary table is created and the data is transferred.

**Step 6: Change the Collation of the Views**

Views can have their own collations, so their collations need to be checked and changed if needed. Having the wrong collation can dramatically change the resulting sort order or even the result itself. For example, joins between Unicode and non-Unicode values using Windows collations give different results than the same joins using SQL Server collations.

To change the collation of a view, you need to drop and re-create the view. Remember to reassign the rights granted to the view afterward.

**Step 7: Rebuild the Indexes**

The last step is to rebuild all indexes using your maintenance plan or a rebuilding job. If you followed my recommendation in step 3, you also need to re-create all the dropped objects at this point.

**A Worthwhile Process**

Using the seven-step process I described here, you can perform a full collation change. As you probably surmised, this isn't a five-minute job. It takes a serious amount of work. However, by following this methodology, you can prevent data loss, incorrect results, unwanted sorting orders, and poor performance.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

****

**Stéphane Haby** is Delivery Manager and Senior Consultant at dbi Services, Switzerland. He has more than 10 years of experience with Microsoft solutions, and he is an MCSA and MCSE for SQL Server 2012 and an MCTS and MCITP for SQL Server 2008.  
**E-mail:** [stephane.haby@dbi-services.com](mailto:stephane.haby@dbi-services.com" \t "_blank)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* [Print](http://sqlmag.com/penton_ur/nojs/user/register?nid=16341&source=print" \o "Display a printer-friendly version of this page.)
* [reprints](http://wrightsmedia.com/sites/penton/it-developer/)
* [Favorite](http://sqlmag.com/user)
  + [EMAIL](http://sqlmag.com/penton_ur/nojs/user/register?path=node%2F16341&nid=16341&source=email" \o "Forward this page)
* [in](javascript:void(0);)**[Share](javascript:void(0);)**
* [Tweet](http://twitter.com/share)

**Discuss this Article 2**

**[ibrahimfahd](http://sqlmag.com/users/ibrahimfahd" \o "View user profile.)**

on Jul 8, 2015

You may simply create an empty new database and choose the collation you want to have, then simply import the data from the original database. If you want to preserve the primary and foreign keys you can generate them and apply them on the new database.

* **[Log In](http://sqlmag.com/penton_ur/nojs/login" \o ") or [Register](http://sqlmag.com/penton_ur/nojs/user/register?source=comment&nid=16341" \o "Register at this site.) to post comments**

**[Dejan](http://sqlmag.com/users/dejan" \o "View user profile.)**

on Nov 16, 2015

Ah! Thank you Ibrahim, I lost all afternoon trying to solve this problem and it was so simple using your method!!