








Collation and Unicode support

Article • 07/13/2023

Applies to:  SQL Server  Azure SQL Database  Azure SQL Managed Instance  Azure Synapse Analytics  Analytics Platform System (PDW)  SQL analytics endpoint in Microsoft Fabric  Warehouse in Microsoft Fabric

Collations in SQL Server provide sorting rules, case, and accent sensitivity properties for your data. Collations that are used with character data types, such as **char** and **varchar**, dictate the code page and corresponding characters that can be represented for that data type.

Whether you're installing a new instance of SQL Server, restoring a database backup, or connecting server to client databases, it's important to understand the locale requirements, sorting order, and case and accent sensitivity of the data that you're working with. To list the collations that are available on your instance of SQL Server, see [sys.fn_helpcollations \(Transact-SQL\)](#).

When you select a collation for your server, database, column, or expression, you're assigning certain characteristics to your data. These characteristics affect the results of many operations in the database. For example, when you construct a query by using `ORDER BY`, the sort order of your result set might depend on the collation that's applied to the database or dictated in a `COLLATE` clause at the expression level of the query.

To best use collation support in SQL Server, you should understand the terms that are defined in this article and how they relate to the characteristics of your data.

Collation terms

- [Collation](#)
 - [Collation sets](#)
 - [Collation levels](#)
- [Locale](#)
- [Code page](#)
- [Sort order](#)

Collation

A collation specifies the bit patterns that represent each character in a dataset. Collations also determine the rules that sort and compare data. SQL Server supports

storing objects that have different collations in a single database. For non-Unicode columns, the collation setting specifies the code page for the data and which characters can be represented. The data that you move between non-Unicode columns must be converted from the source code page to the destination code page.

Transact-SQL statement results can vary when the statement is run in the context of different databases that have different collation settings. If possible, use a standardized collation for your organization. This way, you don't have to specify the collation in every character or Unicode expression. If you must work with objects that have different collation and code page settings, code your queries to consider the rules of collation precedence. For more information, see [Collation Precedence \(Transact-SQL\)](#).


The options associated with a collation are case sensitivity, accent sensitivity, kana sensitivity, width sensitivity, and variation-selector sensitivity. SQL Server 2019 (15.x) introduces an additional option for [UTF-8](#) encoding.

You can specify these options by appending them to the collation name. For example, the collation **Japanese_Bushu_Kakusu_100_CS_AS_KS_WS_UTF8** is case-sensitive, accent-sensitive, kana-sensitive, width-sensitive, and UTF-8 encoded. As another example, the collation **Japanese_Bushu_Kakusu_140_CI_AI_KS_WS_VSS** is case-insensitive, accent-insensitive, kana-sensitive, width-sensitive, variation-selector-sensitive, and it uses non-Unicode encoding.

The behavior associated with these various options is described in the following table:

[Expand table](#)

Option	Description
Case-sensitive (_CS)	Distinguishes between uppercase and lowercase letters. If this option is selected, lowercase letters sort ahead of their uppercase versions. If this option isn't selected, the collation is case-insensitive. That is, SQL Server considers the uppercase and lowercase versions of letters to be identical for sorting purposes. You can explicitly select case insensitivity by specifying _CI.
Accent-sensitive (_AS)	Distinguishes between accented and unaccented characters. For example, "a" isn't equal to "ă". If this option isn't selected, the collation is accent-insensitive. That is, SQL Server considers the accented and unaccented versions of letters to be identical for sorting purposes. You can explicitly select accent insensitivity by specifying _AI.
Kana-sensitive (_KS)	Distinguishes between the two types of Japanese kana characters: Hiragana and Katakana. If this option isn't selected, the collation is kana-insensitive. That is, SQL Server considers Hiragana and Katakana characters to be equal for sorting purposes. Omitting this option is the only method of specifying kana-insensitivity.

Option	Description
Width-sensitive (_WS)	Distinguishes between full-width and half-width characters. If this option isn't selected, SQL Server considers the full-width and half-width representation of the same character to be identical for sorting purposes. Omitting this option is the only method of specifying width-insensitivity.
Variation-selector-sensitive (_VSS)	<p>Distinguishes between various ideographic variation selectors in the Japanese collations Japanese_Bushu_Kakusu_140 and Japanese_XJIS_140, which are introduced in SQL Server 2017 (14.x). A variation sequence consists of a base character plus a variation selector. If this _VSS option isn't selected, the collation is variation-selector-insensitive, and the variation selector isn't considered in the comparison. That is, SQL Server considers characters built upon the same base character with differing variation selectors to be identical for sorting purposes. For more information, see Unicode Ideographic Variation Database .</p> <p>Variation-selector-sensitive (_VSS) collations aren't supported in full-text search indexes. Full-text search indexes support only Accent-Sensitive (_AS), Kana-sensitive (_KS), and Width-sensitive (_WS) options. SQL Server XML and CLR engines don't support (_VSS) Variation selectors.</p>
Binary (_BIN) ¹	Sorts and compares data in SQL Server tables based on the bit patterns defined for each character. Binary sort order is case-sensitive and accent-sensitive. Binary is also the fastest sorting order. For more information, see the Binary collations section in this article.
Binary-code point (_BIN2) ¹	<p>Sorts and compares data in SQL Server tables based on Unicode code points for Unicode data. For non-Unicode data, Binary-code point uses comparisons that are identical to those for binary sorts.</p> <p>The advantage of using a Binary-code point sort order is that no data resorting is required in applications that compare sorted SQL Server data. As a result, a Binary-code point sort order provides simpler application development and possible performance increases. For more information, see the Binary collations section in this article.</p>
UTF-8 (_UTF8)	Enables UTF-8 encoded data to be stored in SQL Server. If this option isn't selected, SQL Server uses the default non-Unicode encoding format for the applicable data types. For more information, see the UTF-8 Support section in this article.

¹ If Binary or Binary-code point is selected, the Case-sensitive (_CS), Accent-sensitive (_AS), Kana-sensitive (_KS), and Width-sensitive (_WS) options aren't available.

Examples of collation options

Each collation is combined as a series of suffixes to define case-, accent-, width-, or kana-sensitivity. The following examples describe sort order behavior for various combinations of suffixes.

Windows collation suffix	Sort order description
_BIN ¹	Binary sort
_BIN2 ^{1, 2}	Binary-code point sort order
_CI_AI ²	Case-insensitive, accent-insensitive, kana-insensitive, width-insensitive
_CI_AI_KS ²	Case-insensitive, accent-insensitive, kana-sensitive, width-insensitive
_CI_AI_KS_WS ²	Case-insensitive, accent-insensitive, kana-sensitive, width-sensitive
_CI_AI_WS ²	Case-insensitive, accent-insensitive, kana-insensitive, width-sensitive
_CI_AS ²	Case-insensitive, accent-sensitive, kana-insensitive, width-insensitive
_CI_AS_KS ²	Case-insensitive, accent-sensitive, kana-sensitive, width-insensitive
_CI_AS_KS_WS ²	Case-insensitive, accent-sensitive, kana-sensitive, width-sensitive
_CI_AS_WS ²	Case-insensitive, accent-sensitive, kana-insensitive, width-sensitive
_CS_AI ²	Case-sensitive, accent-insensitive, kana-insensitive, width-insensitive
_CS_AI_KS ²	Case-sensitive, accent-insensitive, kana-sensitive, width-insensitive
_CS_AI_KS_WS ²	Case-sensitive, accent-insensitive, kana-sensitive, width-sensitive
_CS_AI_WS ²	Case-sensitive, accent-insensitive, kana-insensitive, width-sensitive
_CS_AS ²	Case-sensitive, accent-sensitive, kana-insensitive, width-insensitive
_CS_AS_KS ²	Case-sensitive, accent-sensitive, kana-sensitive, width-insensitive
_CS_AS_KS_WS ²	Case-sensitive, accent-sensitive, kana-sensitive, width-sensitive
_CS_AS_WS ²	Case-sensitive, accent-sensitive, kana-insensitive, width-sensitive

¹ If Binary or Binary-code point is selected, the Case-sensitive (_CS), Accent-sensitive (_AS), Kana-sensitive (_KS), and Width-sensitive (_WS) options aren't available.

² Adding the UTF-8 option (_UTF8) enables you to encode Unicode data by using UTF-8. For more information, see the [UTF-8 Support](#) section in this article.

Collation sets

SQL Server supports the following collation sets:

- [Windows collations](#)
- [Binary collations](#)
- [SQL Server collations](#)

Windows collations

Windows collations define rules for storing character data that's based on an associated Windows system locale. For a Windows collation, you can implement a comparison of non-Unicode data by using the same algorithm as that for Unicode data. The base Windows collation rules specify which alphabet or language is used when dictionary sorting is applied. The rules also specify the code page that's used to store non-Unicode character data. Both Unicode and non-Unicode sorting are compatible with string comparisons in a particular version of Windows. This provides consistency across data types within SQL Server, and it lets developers sort strings in their applications by using the same rules that are used by SQL Server. For more information, see [Windows Collation Name \(Transact-SQL\)](#).

Binary collations

Binary collations sort data based on the sequence of coded values that are defined by the locale and data type. They're case-sensitive. A binary collation in SQL Server defines the locale and the ANSI code page that's used. This enforces a binary sort order. Because they're relatively simple, binary collations help improve application performance. For non-Unicode data types, data comparisons are based on the code points that are defined on the ANSI code page. For Unicode data types, data comparisons are based on the Unicode code points. For binary collations on Unicode data types, the locale isn't considered in data sorts. For example, **Latin1_General_BIN** and **Japanese_BIN** yield identical sorting results when they're used on Unicode data. For more information, see [Windows Collation Name \(Transact-SQL\)](#).

There are two types of binary collations in SQL Server:

- The legacy **BIN** collations, which performed an incomplete code-point-to-code-point comparison for Unicode data. These legacy binary collations compared the first character as WCHAR, followed by a byte-by-byte comparison. In a BIN collation, only the first character is sorted according to the code point, and remaining characters are sorted according to their byte values.
- The newer **BIN2** collations, which implement a pure code-point comparison. In a BIN2 collation, all characters are sorted according to their code points. Because the

Intel platform is a little endian architecture, Unicode code characters are always stored byte-swapped.

SQL Server collations

SQL Server collations (SQL_*) provide sort order compatibility with earlier versions of SQL Server. The dictionary sorting rules for non-Unicode data are incompatible with any sorting routine that's provided by Windows operating systems. However, sorting Unicode data is compatible with a particular version of Windows sorting rules. Because SQL Server collations use different comparison rules for non-Unicode and Unicode data, you see different results for comparisons of the same data, depending on the underlying data type.

For example, if you are using the SQL collation **SQL_Latin1_General_CP1_CI_AS**, the non-Unicode string 'a-c' is less than the string 'ab' because the hyphen (-) is sorted as a separate character that comes before b. However, if you convert these strings to Unicode and you perform the same comparison, the Unicode string N'a-c' is considered to be greater than N'ab', because the Unicode sorting rules use a *word sort* that ignores the hyphen.

For more information, see [SQL Server Collation Name \(Transact-SQL\)](#).

During SQL Server setup, the default installation collation setting is determined by the operating system (OS) locale. You can change the server-level collation either during setup or by changing the OS locale before installation. For backward compatibility reasons, the default collation is set to the oldest available version that's associated with each specific locale. Therefore, this isn't always the recommended collation. To take full advantage of SQL Server features, change the default installation settings to use Windows collations. For example, for the OS locale "English (United States)" (code page 1252), the default collation during setup is **SQL_Latin1_General_CP1_CI_AS**, and it can be changed to its closest Windows collation counterpart, **Latin1_General_100_CI_AS_SC**.

ⓘ Note

When you upgrade an English-language instance of SQL Server, you can specify SQL Server collations (SQL_*) for compatibility with existing instances of SQL Server. Because the default collation for an instance of SQL Server is defined during setup, make sure that you specify the collation settings carefully when the following conditions are true:

- Your application code depends on the behavior of previous SQL Server collations.
- You must store character data that reflects multiple languages.

Collation levels

Setting collations are supported at the following levels of an instance of SQL Server:

- [Server-level collations](#)
- [Database-level collations](#)
- [Column-level collations](#)
- [Expression-level collations](#)

Server-level collations

The default server collation is determined during SQL Server setup, and it becomes the default collation of the system databases and all user databases.

The following table shows the default collation designations, as determined by the operating system (OS) locale, including their Windows and SQL Language Code Identifiers (LCID):

 Expand table

Windows locale	Windows LCID	SQL LCID	Default collation
Afrikaans (South Africa)	0x0436	0x0409	Latin1_General_CI_AS
Albanian (Albania)	0x041c	0x041c	Albanian_CI_AS
Alsatian (France)	0x0484	0x0409	Latin1_General_CI_AS
Amharic (Ethiopia)	0x045e	0x0409	Latin1_General_CI_AS
Arabic (Algeria)	0x1401	0x0401	Arabic_CI_AS
Arabic (Bahrain)	0x3c01	0x0401	Arabic_CI_AS
Arabic (Egypt)	0x0c01	0x0401	Arabic_CI_AS
Arabic (Iraq)	0x0801	0x0401	Arabic_CI_AS
Arabic (Jordan)	0x2c01	0x0401	Arabic_CI_AS

Windows locale	Windows LCID	SQL LCID	Default collation
Arabic (Kuwait)	0x3401	0x0401	Arabic_CI_AS
Arabic (Lebanon)	0x3001	0x0401	Arabic_CI_AS
Arabic (Libya)	0x1001	0x0401	Arabic_CI_AS
Arabic (Morocco)	0x1801	0x0401	Arabic_CI_AS
Arabic (Oman)	0x2001	0x0401	Arabic_CI_AS
Arabic (Qatar)	0x4001	0x0401	Arabic_CI_AS
Arabic (Saudi Arabia)	0x0401	0x0401	Arabic_CI_AS
Arabic (Syria)	0x2801	0x0401	Arabic_CI_AS
Arabic (Tunisia)	0x1c01	0x0401	Arabic_CI_AS
Arabic (U.A.E.)	0x3801	0x0401	Arabic_CI_AS
Arabic (Yemen)	0x2401	0x0401	Arabic_CI_AS
Armenian (Armenia)	0x042b	0x0419	Latin1_General_CI_AS
Assamese (India)	0x044d	0x044d	Not available at server level
Azerbaijani (Azerbaijan, Cyrillic)	0x082c	0x082c	Deprecated, not available at server level
Azerbaijani (Azerbaijan, Latin)	0x042c	0x042c	Deprecated, not available at server level
Bashkir (Russia)	0x046d	0x046d	Latin1_General_CI_AI
Basque (Basque)	0x042d	0x0409	Latin1_General_CI_AS
Belarusian (Belarus)	0x0423	0x0419	Cyrillic_General_CI_AS
Bangla (Bangladesh)	0x0845	0x0445	Not available at server level
Bengali (India)	0x0445	0x0439	Not available at server level
Bosnian (Bosnia and Herzegovina, Cyrillic)	0x201a	0x201a	Latin1_General_CI_AI
Bosnian (Bosnia and Herzegovina, Latin)	0x141a	0x141a	Latin1_General_CI_AI
Breton (France)	0x047e	0x047e	Latin1_General_CI_AI

Windows locale	Windows LCID	SQL LCID	Default collation
Bulgarian (Bulgaria)	0x0402	0x0419	Cyrillic_General_CI_AS
Catalan (Catalan)	0x0403	0x0409	Latin1_General_CI_AS
Chinese (Hong Kong SAR, PRC)	0x0c04	0x0404	Chinese_Taiwan_Stroke_CI_AS
Chinese (Macao SAR)	0x1404	0x1404	Latin1_General_CI_AI
Chinese (Macao SAR)	0x21404	0x21404	Latin1_General_CI_AI
Chinese (PRC)	0x0804	0x0804	Chinese_PRC_CI_AS
Chinese (PRC)	0x20804	0x20804	Chinese_PRC_Stroke_CI_AS
Chinese (Singapore)	0x1004	0x0804	Chinese_PRC_CI_AS
Chinese (Singapore)	0x21004	0x20804	Chinese_PRC_Stroke_CI_AS
Chinese (Taiwan)	0x30404	0x30404	Chinese_Taiwan_Bopomofo_CI_AS
Chinese (Taiwan)	0x0404	0x0404	Chinese_Taiwan_Stroke_CI_AS
Corsican (France)	0x0483	0x0483	Latin1_General_CI_AI
Croatian (Bosnia and Herzegovina, Latin)	0x101a	0x041a	Croatian_CI_AS
Croatian (Croatia)	0x041a	0x041a	Croatian_CI_AS
Czech (Czech Republic)	0x0405	0x0405	Czech_CI_AS
Danish (Denmark)	0x0406	0x0406	Danish_Norwegian_CI_AS
Dari (Afghanistan)	0x048c	0x048c	Latin1_General_CI_AI
Divehi (Maldives)	0x0465	0x0465	Not available at server level
Dutch (Belgium)	0x0813	0x0409	Latin1_General_CI_AS
Dutch (Netherlands)	0x0413	0x0409	Latin1_General_CI_AS
English (Australia)	0x0c09	0x0409	Latin1_General_CI_AS
English (Belize)	0x2809	0x0409	Latin1_General_CI_AS
English (Canada)	0x1009	0x0409	Latin1_General_CI_AS
English (Caribbean)	0x2409	0x0409	Latin1_General_CI_AS
English (India)	0x4009	0x0409	Latin1_General_CI_AS

Windows locale	Windows LCID	SQL LCID	Default collation
English (Ireland)	0x1809	0x0409	Latin1_General_CI_AS
English (Jamaica)	0x2009	0x0409	Latin1_General_CI_AS
English (Malaysia)	0x4409	0x0409	Latin1_General_CI_AS
English (New Zealand)	0x1409	0x0409	Latin1_General_CI_AS
English (Philippines)	0x3409	0x0409	Latin1_General_CI_AS
English (Singapore)	0x4809	0x0409	Latin1_General_CI_AS
English (South Africa)	0x1c09	0x0409	Latin1_General_CI_AS
English (Trinidad and Tobago)	0x2c09	0x0409	Latin1_General_CI_AS
English (United Kingdom)	0x0809	0x0409	Latin1_General_CI_AS
English (United States)	0x0409	0x0409	SQL_Latin1_General_CP1_CI_AS
English (Zimbabwe)	0x3009	0x0409	Latin1_General_CI_AS
Estonian (Estonia)	0x0425	0x0425	Estonian_CI_AS
Faroese (Faroe Islands)	0x0438	0x0409	Latin1_General_CI_AS
Filipino (Philippines)	0x0464	0x0409	Latin1_General_CI_AS
Finnish (Finland)	0x040b	0x040b	Finnish_Swedish_CI_AS
French (Belgium)	0x080c	0x040c	French_CI_AS
French (Canada)	0x0c0c	0x040c	French_CI_AS
French (France)	0x040c	0x040c	French_CI_AS
French (Luxembourg)	0x140c	0x040c	French_CI_AS
French (Monaco)	0x180c	0x040c	French_CI_AS
French (Switzerland)	0x100c	0x040c	French_CI_AS
Frisian (Netherlands)	0x0462	0x0462	Latin1_General_CI_AI
Galician	0x0456	0x0409	Latin1_General_CI_AS
Georgian (Georgia)	0x10437	0x10437	Georgian_Modern_Sort_CI_AS
Georgian (Georgia)	0x0437	0x0419	Latin1_General_CI_AS

Windows locale	Windows LCID	SQL LCID	Default collation
German - Phone Book Sort (DIN)	0x10407	0x10407	German_PhoneBook_CI_AS
German (Austria)	0x0c07	0x0409	Latin1_General_CI_AS
German (Germany)	0x0407	0x0409	Latin1_General_CI_AS
German (Liechtenstein)	0x1407	0x0409	Latin1_General_CI_AS
German (Luxembourg)	0x1007	0x0409	Latin1_General_CI_AS
German (Switzerland)	0x0807	0x0409	Latin1_General_CI_AS
Greek (Greece)	0x0408	0x0408	Greek_CI_AS
Greenlandic (Greenland)	0x046f	0x0406	Danish_Norwegian_CI_AS
Gujarati (India)	0x0447	0x0439	Not available at server level
Hausa (Nigeria, Latin)	0x0468	0x0409	Latin1_General_CI_AS
Hebrew (Israel)	0x040d	0x040d	Hebrew_CI_AS
Hindi (India)	0x0439	0x0439	Not available at server level
Hungarian (Hungary)	0x040e	0x040e	Hungarian_CI_AS
Hungarian Technical Sort	0x1040e	0x1040e	Hungarian_Technical_CI_AS
Icelandic (Iceland)	0x040f	0x040f	Icelandic_CI_AS
Igbo (Nigeria)	0x0470	0x0409	Latin1_General_CI_AS
Indonesian (Indonesia)	0x0421	0x0409	Latin1_General_CI_AS
Inuktitut (Canada, Latin)	0x085d	0x0409	Latin1_General_CI_AS
Inuktitut (Syllabics) Canada	0x045d	0x045d	Latin1_General_CI_AI
Irish (Ireland)	0x083c	0x0409	Latin1_General_CI_AS
Italian (Italy)	0x0410	0x0409	Latin1_General_CI_AS
Italian (Switzerland)	0x0810	0x0409	Latin1_General_CI_AS
Japanese (Japan XJIS)	0x0411	0x0411	Japanese_CI_AS
Japanese (Japan)	0x040411	0x40411	Latin1_General_CI_AI
Kannada (India)	0x044b	0x0439	Not available at server level

Windows locale	Windows LCID	SQL LCID	Default collation
Kazakh (Kazakhstan)	0x043f	0x043f	Kazakh_90_CI_AS
Khmer (Cambodia)	0x0453	0x0453	Not available at server level
K'iche (Guatemala)	0x0486	0x0c0a	Modern_Spanish_CI_AS
Kinyarwanda (Rwanda)	0x0487	0x0409	Latin1_General_CI_AS
Konkani (India)	0x0457	0x0439	Not available at server level
Korean (Korea Dictionary Sort)	0x0412	0x0412	Korean_Wansung_CI_AS
Kyrgyz (Kyrgyzstan)	0x0440	0x0419	Cyrillic_General_CI_AS
Lao (Lao PDR)	0x0454	0x0454	Not available at server level
Latvian (Latvia)	0x0426	0x0426	Latvian_CI_AS
Lithuanian (Lithuania)	0x0427	0x0427	Lithuanian_CI_AS
Lower Sorbian (Germany)	0x082e	0x0409	Latin1_General_CI_AS
Luxembourgish (Luxembourg)	0x046e	0x0409	Latin1_General_CI_AS
Macedonian (North Macedonia)	0x042f	0x042f	Macedonian_FYROM_90_CI_AS
Malay (Brunei Darussalam)	0x083e	0x0409	Latin1_General_CI_AS
Malay (Malaysia)	0x043e	0x0409	Latin1_General_CI_AS
Malayalam (India)	0x044c	0x0439	Not available at server level
Maltese (Malta)	0x043a	0x043a	Latin1_General_CI_AI
Maori (New Zealand)	0x0481	0x0481	Latin1_General_CI_AI
Mapudungun (Chile)	0x047a	0x047a	Latin1_General_CI_AI
Marathi (India)	0x044e	0x0439	Not available at server level
Mohawk (Canada)	0x047c	0x047c	Latin1_General_CI_AI
Mongolian (Mongolia)	0x0450	0x0419	Cyrillic_General_CI_AS
Mongolian (PRC)	0x0850	0x0419	Cyrillic_General_CI_AS
Nepali (Nepal)	0x0461	0x0461	Not available at server level
Norwegian (Bokmål, Norway)	0x0414	0x0414	Latin1_General_CI_AI
Norwegian (Nynorsk, Norway)	0x0814	0x0414	Latin1_General_CI_AI

Windows locale	Windows LCID	SQL LCID	Default collation
Occitan (France)	0x0482	0x040c	French_CI_AS
Odia (India)	0x0448	0x0439	Not available at server level
Pashto (Afghanistan)	0x0463	0x0463	Not available at server level
Persian (Iran)	0x0429	0x0429	Latin1_General_CI_AI
Polish (Poland)	0x0415	0x0415	Polish_CI_AS
Portuguese (Brazil)	0x0416	0x0409	Latin1_General_CI_AS
Portuguese (Portugal)	0x0816	0x0409	Latin1_General_CI_AS
Punjabi (India)	0x0446	0x0439	Not available at server level
Quechua (Bolivia)	0x046b	0x0409	Latin1_General_CI_AS
Quechua (Ecuador)	0x086b	0x0409	Latin1_General_CI_AS
Quechua (Peru)	0x0c6b	0x0409	Latin1_General_CI_AS
Romanian (Romania)	0x0418	0x0418	Romanian_CI_AS
Romansh (Switzerland)	0x0417	0x0417	Latin1_General_CI_AI
Russian (Russia)	0x0419	0x0419	Cyrillic_General_CI_AS
Sahka (Russia)	0x0485	0x0485	Latin1_General_CI_AI
Sami (Inari, Finland)	0x243b	0x083b	Latin1_General_CI_AI
Sami (Lule, Norway)	0x103b	0x043b	Latin1_General_CI_AI
Sami (Lule, Sweden)	0x143b	0x083b	Latin1_General_CI_AI
Sami (Northern, Finland)	0x0c3b	0x083b	Latin1_General_CI_AI
Sami (Northern, Norway)	0x043b	0x043b	Latin1_General_CI_AI
Sami (Northern, Sweden)	0x083b	0x083b	Latin1_General_CI_AI
Sami (Skolt, Finland)	0x203b	0x083b	Latin1_General_CI_AI
Sami (Southern, Norway)	0x183b	0x043b	Latin1_General_CI_AI
Sami (Southern, Sweden)	0x1c3b	0x083b	Latin1_General_CI_AI
Sanskrit (India)	0x044f	0x0439	Not available at server level

Windows locale	Windows LCID	SQL LCID	Default collation
Serbian (Bosnia and Herzegovina, Cyrillic)	0x1c1a	0x0c1a	Latin1_General_CI_AI
Serbian (Bosnia and Herzegovina, Latin)	0x181a	0x081a	Latin1_General_CI_AI
Serbian (Serbia, Cyrillic)	0x0c1a	0x0c1a	Latin1_General_CI_AI
Serbian (Serbia, Latin)	0x081a	0x081a	Latin1_General_CI_AI
Sesotho sa Leboa/Northern Sotho (South Africa)	0x046c	0x0409	Latin1_General_CI_AS
Setswana/Tswana (South Africa)	0x0432	0x0409	Latin1_General_CI_AS
Sinhala (Sri Lanka)	0x045b	0x0439	Not available at server level
Slovak (Slovakia)	0x041b	0x041b	Slovak_CI_AS
Slovenian (Slovenia)	0x0424	0x0424	Slovenian_CI_AS
Spanish (Argentina)	0x2c0a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Bolivia)	0x400a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Chile)	0x340a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Colombia)	0x240a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Costa Rica)	0x140a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Dominican Republic)	0x1c0a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Ecuador)	0x300a	0x0c0a	Modern_Spanish_CI_AS
Spanish (El Salvador)	0x440a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Guatemala)	0x100a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Honduras)	0x480a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Mexico)	0x080a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Nicaragua)	0x4c0a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Panama)	0x180a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Paraguay)	0x3c0a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Peru)	0x280a	0x0c0a	Modern_Spanish_CI_AS

Windows locale	Windows LCID	SQL LCID	Default collation
Spanish (Puerto Rico)	0x500a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Spain)	0x0c0a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Spain, Traditional Sort)	0x040a	0x040a	Traditional_Spanish_CI_AS
Spanish (United States)	0x540a	0x0409	Latin1_General_CI_AS
Spanish (Uruguay)	0x380a	0x0c0a	Modern_Spanish_CI_AS
Spanish (Venezuela)	0x200a	0x0c0a	Modern_Spanish_CI_AS
Swahili (Kenya)	0x0441	0x0409	Latin1_General_CI_AS
Swedish (Finland)	0x081d	0x040b	Finnish_Swedish_CI_AS
Swedish (Sweden)	0x041d	0x040b	Finnish_Swedish_CI_AS
Syriac (Syria)	0x045a	0x045a	Not available at server level
Tajik (Tajikistan)	0x0428	0x0419	Cyrillic_General_CI_AS
Tamazight (Algeria, Latin)	0x085f	0x085f	Latin1_General_CI_AI
Tamil (India)	0x0449	0x0439	Not available at server level
Tatar (Russia)	0x0444	0x0444	Cyrillic_General_CI_AS
Telugu (India)	0x044a	0x0439	Not available at server level
Thai (Thailand)	0x041e	0x041e	Thai_CI_AS
Tibetan (PRC)	0x0451	0x0451	Not available at server level
Turkish (Türkiye)	0x041f	0x041f	Turkish_CI_AS
Turkmen (Turkmenistan)	0x0442	0x0442	Latin1_General_CI_AI
Uighur (PRC)	0x0480	0x0480	Latin1_General_CI_AI
Ukrainian (Ukraine)	0x0422	0x0422	Ukrainian_CI_AS
Upper Sorbian (Germany)	0x042e	0x042e	Latin1_General_CI_AI
Urdu (Pakistan)	0x0420	0x0420	Latin1_General_CI_AI
Uzbek (Uzbekistan, Cyrillic)	0x0843	0x0419	Cyrillic_General_CI_AS
Uzbek (Uzbekistan, Latin)	0x0443	0x0443	Uzbek_Latin_90_CI_AS
Vietnamese (Vietnam)	0x042a	0x042a	Vietnamese_CI_AS

Windows locale	Windows LCID	SQL LCID	Default collation
Welsh (United Kingdom)	0x0452	0x0452	Latin1_General_CI_AI
Wolof (Senegal)	0x0488	0x040c	French_CI_AS
Xhosa/isiXhosa (South Africa)	0x0434	0x0409	Latin1_General_CI_AS
Yi (PRC)	0x0478	0x0409	Latin1_General_CI_AS
Yoruba (Nigeria)	0x046a	0x0409	Latin1_General_CI_AS
Zulu/isiZulu (South Africa)	0x0435	0x0409	Latin1_General_CI_AS

After you've assigned a collation to the server, you can change it only by exporting all database objects and data, rebuilding the `master` database, and importing all database objects and data. Instead of changing the default collation of an instance of SQL Server, you can specify the desired collation when you create a new database or database column.

To query the server collation for an instance of SQL Server, use the `SERVERPROPERTY` function:

SQL

```
SELECT CONVERT(nvarchar(128), SERVERPROPERTY('collation'));
```

To query the server for all available collations, use the following `fn_helpcollations()` built-in function:

SQL

```
SELECT * FROM sys.fn_helpcollations();
```

Collations in Azure SQL Database

You cannot change or set the logical server collation on Azure SQL Database, but can configure each database's collations both for data and for the catalog. The catalog collation determines the collation for system metadata, such as object identifiers. Both collations can be specified independently when you [create the database in the Azure portal](#), in T-SQL with `CREATE DATABASE`, in PowerShell with `New-AzSqlDatabase`.

Collations in Azure SQL Managed Instance

Server-level collation in Azure SQL Managed Instance can be specified when the instance is created and cannot be changed later.

For more information, see [Set or Change the Server Collation](#).

Database-level collations

When you create or modify a database, you can use the `COLLATE` clause of the `CREATE DATABASE` or `ALTER DATABASE` statement to specify the default database collation. If no collation is specified, the database is assigned the server collation.

You can't change the collation of system databases unless you change the collation for the server.

- In SQL Server and Azure SQL Managed Instance, the database collation is used for all metadata in the database, and the collation is the default for all string columns, temporary objects, variable names, and any other strings used in the database.
- In Azure SQL Database, there is no server collation, so each database has a collation for data and a collation for the catalog. The `CATALOG_COLLATION` is used for all metadata in the database, and the collation is the default for all string columns, temporary objects, variable names, and any other strings used in the database. The `CATALOG_COLLATION` is set upon creation and cannot be changed.

When you change the collation of a user database, there can be collation conflicts when queries in the database access temporary tables. Temporary tables are always stored in the `tempdb` system database, which uses the collation for the instance. Queries that compare character data between the user database and `tempdb` might fail if the collations cause a conflict in evaluating the character data. You can resolve this issue by specifying the `COLLATE` clause in the query. For more information, see [COLLATE \(Transact-SQL\)](#).

You can change the collation of a user database by using an `ALTER DATABASE` statement similar to the following code sample:

SQL

```
ALTER DATABASE myDB COLLATE Greek_CS_AI;
```

 **Important**

Altering the database-level collation doesn't affect column-level or expression-level collations. It does not affect data in existing columns.

You can retrieve the current collation of a database by using a statement similar to the following code sample:

SQL

```
SELECT CONVERT (nvarchar(128), DATABASEPROPERTYEX('database_name',  
'collation'));
```

Column-level collations

When you create or alter a table, you can specify collations for each character-string column by using the `COLLATE` clause. If you don't specify a collation, the column is assigned the default collation of the database.

You can change the collation of a column by using an `ALTER TABLE` statement similar to the following code sample:

SQL

```
ALTER TABLE myTable ALTER COLUMN mycol NVARCHAR(10) COLLATE  
Greek_CS_AI;
```

Expression-level collations

Expression-level collations are set when a statement is run, and they affect the way a result set is returned. This enables `ORDER BY` sort results to be locale-specific. To implement expression-level collations, use a `COLLATE` clause such as the following code sample:

SQL

```
SELECT name FROM customer ORDER BY name COLLATE Latin1_General_CS_AI;
```

Locale

A locale is a set of information that's associated with a location or a culture. The information can include the name and identifier of the spoken language, the script

that's used to write the language, and cultural conventions. Collations can be associated with one or more locales. For more information, see [Locale IDs Assigned by Microsoft](#).

Code page

A code page is an ordered set of characters of a given script in which a numeric index, or code point value, is associated with each character. A Windows code page is typically referred to as a *character set* or a *charset*. Code pages are used to provide support for the character sets and keyboard layouts that are used by different Windows system locales.

Sort order

Sort order specifies how data values are sorted. The order affects the results of data comparison. Data is sorted by using collations, and it can be optimized by using indexes.

Unicode support

Unicode is a standard for mapping code points to characters. Because it's designed to cover all the characters of all the languages of the world, you don't need different code pages to handle different sets of characters.

Unicode basics

Storing data in multiple languages within one database is difficult to manage when you use only character data and code pages. It's also difficult to find one code page for the database that can store all the required language-specific characters. Additionally, it's difficult to guarantee the correct translation of special characters when they're being read or updated by a variety of clients that are running various code pages. Databases that support international clients should always use Unicode data types instead of non-Unicode data types.

For example, consider a database of customers in North America that must handle three major languages:

- Spanish names and addresses for Mexico
- French names and addresses for Quebec
- English names and addresses for the rest of Canada and the United States

When you use only character columns and code pages, you must take care to ensure that the database is installed with a code page that will handle the characters of all three languages. You must also take care to guarantee the correct translation of characters from any of the languages when the characters are read by clients that are running a code page for another language.

ⓘ Note

The code pages that a client uses are determined by the operating system (OS) settings. To set client code pages on the Windows operating system, use **Regional Settings** in Control Panel.

It would be difficult to select a code page for character data types that will support all the characters that are required by a worldwide audience. The easiest way to manage character data in international databases is to always use a data type that supports Unicode.

Unicode data types

If you store character data that reflects multiple languages in SQL Server (SQL Server 2005 (9.x) and later), use Unicode data types (**nchar**, **nvarchar**, and **ntext**) instead of non-Unicode data types (**char**, **varchar**, and **text**).

ⓘ Note

For Unicode data types, the Database Engine can represent up to 65,536 characters using UCS-2, or the full Unicode range (1,114,112 characters) if supplementary characters are used. For more information about enabling supplementary characters, see [Supplementary Characters](#).

Alternatively, starting with SQL Server 2019 (15.x), if a UTF-8 enabled collation (**_UTF8**) is used, previously non-Unicode data types (**char** and **varchar**) become Unicode data types using UTF-8 encoding. SQL Server 2019 (15.x) doesn't change the behavior of previously existing Unicode data types (**nchar**, **nvarchar**, and **ntext**), which continue to use UCS-2 or UTF-16 encoding. For more information, see [Storage differences between UTF-8 and UTF-16](#).

Unicode considerations

Significant limitations are associated with non-Unicode data types. This is because a non-Unicode computer is limited to using a single code page. You might experience performance gain by using Unicode, because it requires fewer code-page conversions. Unicode collations must be selected individually at the database, column, or expression level because they aren't supported at the server level.

When you move data from a server to a client, your server collation might not be recognized by older client drivers. This can occur when you move data from a Unicode server to a non-Unicode client. Your best option might be to upgrade the client operating system so that the underlying system collations are updated. If the client has database client software installed, you might consider applying a service update to the database client software.

Tip

You can also try to use a different collation for the data on the server. Choose a collation that maps to a code page on the client.

To use the UTF-16 collations that are available in SQL Server (SQL Server 2012 (11.x) and later) to improve searching and sorting of some Unicode characters (Windows collations only), you can select either one of the supplementary characters (_SC) collations or one of the version 140 collations.

To use the UTF-8 collations that are available in SQL Server 2019 (15.x), and to improve searching and sorting of some Unicode characters (Windows collations only), you must select UTF-8 encoding-enabled collations(_UTF8).

- The UTF8 flag can be applied to:
 - Linguistic collations that already support supplementary characters (_SC) or variation-selector-sensitive (_VSS) awareness
 - BIN2 binary collation
- The UTF8 flag can't be applied to:
 - Linguistic collations that don't support supplementary characters (_SC) or variation-selector-sensitive (_VSS) awareness
 - The BIN binary collations
 - The SQL_* collations

To evaluate issues that are related to using Unicode or non-Unicode data types, test your scenario to measure performance differences in your environment. It's a good practice to standardize the collation that's used on systems across your organization, and to deploy Unicode servers and clients wherever possible.

In many situations, SQL Server interacts with other servers or clients, and your organization might use multiple data-access standards between applications and server instances. SQL Server clients are one of two main types:

- **Unicode clients** that use OLE DB and Open Database Connectivity (ODBC) version 3.7 or later.
- **Non-Unicode clients** that use DB-Library and ODBC version 3.6 or earlier.

The following table provides information about using multilingual data with various combinations of Unicode and non-Unicode servers:

 Expand table

Server	Client	Benefits or limitations
Unicode	Unicode	Because Unicode data is used throughout the system, this scenario provides the best performance and protection from corruption of retrieved data. This is the situation with ActiveX Data Objects (ADO), OLE DB, and ODBC version 3.7 or later.
Unicode	Non-Unicode	In this scenario, especially with connections between a server that's running a newer operating system and a client that's running an earlier version of SQL Server, or on an older operating system, there can be limitations or errors when you move data to a client computer. Unicode data on the server tries to map to a corresponding code page on the non-Unicode client to convert the data.
Non-Unicode	Unicode	This isn't an ideal configuration for using multilingual data. You can't write Unicode data to the non-Unicode server. Problems are likely to occur when data is sent to servers that are outside the server's code page.
Non-Unicode	Non-Unicode	This is a very limiting scenario for multilingual data. You can use only a single code page.

Supplementary characters

The Unicode Consortium allocates to each character a unique code point, which is a value in the range 000000–10FFFF. The most frequently used characters have code point values in the range 000000–00FFFF (65,536 characters) which fit into an 8-bit or 16-bit word in memory and on-disk. This range is usually designated as the Basic Multilingual Plane (BMP).

But the Unicode Consortium has established 16 additional "planes" of characters, each the same size as the BMP. This definition allows Unicode the potential to represent 1,114,112 characters (that is, $2^{16} * 17$ characters) within the code point range 000000–

10FFFF. Characters with code point value larger than 00FFFF require two to four consecutive 8-bit words (UTF-8), or two consecutive 16-bit words (UTF-16). These characters located beyond the BMP are called *supplementary characters*, and the additional consecutive 8-bit or 16-bit words are called *surrogate pairs*. For more information about supplementary characters, surrogates, and surrogate pairs, see [the Unicode Standard](#).

SQL Server provides data types such as **nchar** and **nvarchar** to store Unicode data in the BMP range (000000–00FFFF), which the Database Engine encodes using UCS-2.

SQL Server 2012 (11.x) introduced a new family of supplementary character (_SC) collations that can be used with the **nchar**, **nvarchar**, and **sql_variant** data types to represent the full Unicode character range (000000–10FFFF). For example:

Latin1_General_100_CI_AS_SC or, if you're using a Japanese collation, **Japanese_Bushu_Kakusu_100_CI_AS_SC**.

SQL Server 2019 (15.x) extends supplementary character support to the **char** and **varchar** data types with the new UTF-8 enabled collations (**_UTF8**). These data types are also capable of representing the full Unicode character range.

Note

Starting with SQL Server 2017 (14.x), all new collations automatically support supplementary characters.

If you use supplementary characters:

- Supplementary characters can be used in ordering and comparison operations in collation versions 90 or greater.
- All version 100 collations support linguistic sorting with supplementary characters.
- Supplementary characters aren't supported for use in metadata, such as in names of database objects.
- The SC flag can be applied to:
 - Version 90 collations
 - Version 100 collations
- The SC flag can't be applied to:
 - Version 80 non-versioned Windows collations
 - The BIN or BIN2 binary collations
 - The SQL* collations

- Version 140 collations (these don't need the SC flag, because they already support supplementary characters)

The following table compares the behavior of some string functions and string operators when they use supplementary characters with and without a supplementary character-aware (SCA) collation:

[Expand table](#)

String function or operator	With an SCA collation	Without an SCA collation
CHARINDEX LEN PATINDEX	The UTF-16 surrogate pair is counted as a single code point.	The UTF-16 surrogate pair is counted as two code points.
LEFT REPLACE REVERSE RIGHT SUBSTRING STUFF	These functions treat each surrogate pair as a single code point and work as expected.	These functions might split any surrogate pairs and lead to unexpected results.
NCHAR	Returns the character that corresponds to the specified Unicode code point value in the range 0–0x10FFFF. If the specified value lies in the range 0–0xFFFF, one character is returned. For higher values, the corresponding surrogate is returned.	A value higher than 0xFFFF returns NULL instead of the corresponding surrogate.
UNICODE	Returns a UTF-16 code point in the range 0–0x10FFFF.	Returns a UCS-2 code point in the range 0–0xFFFF.
Match One Character Wildcard Wildcard - Character(s) Not to Match	Supplementary characters are supported for all wildcard operations.	Supplementary characters aren't supported for these wildcard operations. Other wildcard operators are supported.

GB18030 support

GB18030 is a separate standard that's used in the People's Republic of China for encoding Chinese characters. In GB18030, characters can be 1, 2, or 4 bytes in length. SQL Server provides support for GB18030-encoded characters by recognizing them when they enter the server from a client-side application and converting and storing them natively as Unicode characters. After they're stored in the server, they're treated as Unicode characters in any subsequent operations.

You can use any Chinese collation, preferably the latest 100 version. All version 100 collations support linguistic sorting with GB18030 characters. If the data includes supplementary characters (surrogate pairs), you can use the SC collations that are available in SQL Server to improve searching and sorting.

ⓘ Note

Ensure that your client tools, such as SQL Server Management Studio, use the Dengxian font to correctly display strings that contain GB18030-encoded characters.

Complex script support

SQL Server can support inputting, storing, changing, and displaying complex scripts. Complex scripts include the following types:

- Scripts that include the combination of both right-to-left and left-to-right text, such as a combination of Arabic and English text.
- Scripts whose characters change shape depending on their position, or when combined with other characters, such as Arabic, Indic, and Thai characters.
- Languages, such as Thai, that require internal dictionaries to recognize words because there are no breaks between them.

Database applications that interact with SQL Server must use controls that support complex scripts. Standard Windows form controls that are created in managed code are complex-script-enabled.

Japanese collations added in SQL Server 2017 (14.x)

Starting with SQL Server 2017 (14.x), new Japanese collation families are supported, with the permutations of various options (`_CS`, `_AS`, `_KS`, `_WS`, and `_VSS`), as well as `_BIN` and `_BIN2`.

To list these collations, you can query the SQL Server Database Engine:

SQL

```
SELECT name, description
FROM sys.fn_helpcollations()
WHERE COLLATIONPROPERTY(name, 'Version') = 3;
```

All the new collations have built-in support for supplementary characters, so none of the new **140** collations has (or needs) the SC flag.

These collations are supported in Database Engine indexes, memory-optimized tables, columnstore indexes, and natively compiled modules.

UTF-8 support

SQL Server 2019 (15.x) introduces full support for the widely used UTF-8 character encoding as an import or export encoding, and as database-level or column-level collation for string data. UTF-8 is allowed in the **char** and **varchar** data types, and it's enabled when you create or change an object's collation to a collation that has a *UTF8* suffix. One example is changing **LATIN1_GENERAL_100_CI_AS_SC** to **LATIN1_GENERAL_100_CI_AS_SC_UTF8**.

UTF-8 is available only to Windows collations that support supplementary characters, as introduced in SQL Server 2012 (11.x). The **nchar** and **nvarchar** data types allow UCS-2 or UTF-16 encoding only, and they remain unchanged.

Azure SQL Database and Azure SQL Managed Instance also support UTF-8 on database and column level, while SQL Managed Instance supports this on a server level as well.

Storage differences between UTF-8 and UTF-16

The Unicode Consortium allocates to each character a unique code point, which is a value in the range 000000–10FFFF. With SQL Server 2019 (15.x), both UTF-8 and UTF-16 encodings are available to represent the full range:

- With UTF-8 encoding, characters in the ASCII range (000000–00007F) require 1 byte, code points 000080–0007FF require 2 bytes, code points 000800–00FFFF require 3 bytes, and code points 0010000–0010FFFF require 4 bytes.

- With UTF-16 encoding, code points 000000–00FFFF require 2 bytes, and code points 0010000–0010FFFF require 4 bytes.

The following table lists the encoding storage bytes for each character range and encoding type:

[Expand table](#)

Code range (hexadecimal)	Code range (decimal)	Storage bytes ¹ with UTF-8	Storage bytes ¹ with UTF-16
000000–00007F	0–127	1	2
000080–00009F 0000A0–0003FF 000400–0007FF	128–159 160–1,023 1,024–2,047	2	2
000800–003FFF 004000–00FFFF	2,048–16,383 16,384–65,535	3	2
010000–03FFFF ²	65,536–262,143 ²	4	4
040000–10FFFF ²	262,144–1,114,111 ²		

¹ *Storage bytes* refer to the encoded byte length, not the data-type on-disk storage size. For more information about on-disk storage sizes, see [nchar and nvarchar](#) and [char and varchar](#).

² The code point range for [supplementary characters](#).

💡 Tip

It's common to think, in [CHAR\(*n*\) and VARCHAR\(*n*\)](#) or in [NCHAR\(*n*\) and NVARCHAR\(*n*\)](#), that *n* defines the number of characters. This is because, in the example of a CHAR(10) column, 10 ASCII characters in the range 0–127 can be stored by using a collation such as [Latin1_General_100_CI_AI](#), because each character in this range uses only 1 byte.

However, in [CHAR\(*n*\) and VARCHAR\(*n*\)](#), *n* defines the string size in *bytes* (0–8,000), and in [NCHAR\(*n*\) and NVARCHAR\(*n*\)](#), *n* defines the string size in *byte-pairs* (0–4,000). *n* never defines numbers of characters that can be stored.

As you've just seen, choosing the appropriate Unicode encoding and data type might give you significant storage savings or increase your current storage footprint, depending on the character set in use. For example, when you use a Latin collation

that's UTF-8 enabled, such as `Latin1_General_100_CI_AI_SC_UTF8`, a `CHAR(10)` column stores 10 bytes and can hold 10 ASCII characters in the range 0–127. But it can hold only five characters in the range 128–2047 and only three characters in the range 2048–65535. By comparison, because a `NCHAR(10)` column stores 10 byte-pairs (20 bytes), it can hold 10 characters in the range 0–65535.

Before you choose whether to use UTF-8 or UTF-16 encoding for a database or column, consider the distribution of string data that will be stored:

- If it's mostly in the ASCII range 0–127 (such as English), each character requires 1 byte with UTF-8 and 2 bytes with UTF-16. Using UTF-8 provides storage benefits. Changing an existing column data type with ASCII characters in the range 0–127 from `NCHAR(10)` to `CHAR(10)`, and using an UTF-8 enabled collation, translates into a 50 percent reduction in storage requirements. This reduction is because `NCHAR(10)` requires 20 bytes for storage, compared with `CHAR(10)`, which requires 10 bytes for the same Unicode string representation.
- Above the ASCII range, almost all Latin-based script, and Greek, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac, Tāna, and N'Ko, require 2 bytes per character in both UTF-8 and UTF-16. In these cases, there aren't significant storage differences for comparable data types (for example, between using `char` or `nchar`).
- If it's mostly East Asian script (such as Korean, Chinese, and Japanese), each character requires 3 bytes with UTF-8 and 2 bytes with UTF-16. Using UTF-16 provides storage benefits.
- Characters in the range 010000–10FFFF require 4 bytes in both UTF-8 and UTF-16. In these cases, there aren't storage differences for comparable data types (for example, between using `char` or `nchar`).

For other considerations, see [Write International Transact-SQL Statements](#).

Convert to UTF-8

Because in `CHAR(n)` and `VARCHAR(n)` or in `NCHAR(n)` and `NVARCHAR(n)`, the *n* defines the byte storage size, not the number of characters that can be stored, it's important to determine the data type size you must convert to, in order to avoid data truncation.

For example, consider a column defined as `NVARCHAR(100)` that stores 180 bytes of Japanese characters. In this example, the column data is currently encoded using UCS-2 or UTF-16, which uses 2 bytes per character. Converting the column type to `VARCHAR(200)` isn't enough to prevent data truncation, because the new data type can only store 200 bytes, but Japanese characters require 3 bytes when encoded in UTF-8. So the column must be defined as `VARCHAR(270)` to avoid data loss through data truncation.

Therefore, it's required to know in advance what's the projected byte size for the column definition before converting existing data to UTF-8, and adjust the new data type size accordingly. Refer to the Transact-SQL script or the SQL Notebook in the [Data Samples GitHub](#) , which use the `DATALENGTH` function and the `COLLATE` statement to determine the correct data length requirements for UTF-8 conversion operations in an existing database.

To change the column collation and data type in an existing table, use one of the methods described in [Set or Change the Column Collation](#).

To change the database collation, allowing new objects to inherit the database collation by default, or to change the server collation, allowing new databases to inherit the system collation by default, see the [Related tasks](#) section of this article.






Related tasks

 Expand table

Task	Article
Describes how to set or change the collation of the instance of SQL Server. Changing the server collation doesn't change the collation of existing databases.	Set or Change the Server Collation
Describes how to set or change the collation of a user database. Changing a database collation doesn't change the collation of existing table columns.	Set or Change the Database Collation
Describes how to set or change the collation of a column in the database	Set or Change the Column Collation
Describes how to return collation information at the server, database, or column level	View Collation Information
Describes how to write Transact-SQL statements that are more portable from one language to another, or support multiple languages more easily	Write International Transact-SQL Statements
Describes how to change the language of error messages and preferences for how date, time, and currency data are used and displayed	Set a Session Language

Related content

For more information, see the following related content:

- [SQL Server Best Practices Collation Change](#) 
- [Use Unicode Character Format to Import or Export Data \(SQL Server\)](#)
- [Write International Transact-SQL Statements](#)
- [SQL Server Best Practices Migration to Unicode](#)  (no longer maintained)
- [Unicode Consortium](#) 
- [Unicode Standard](#) 
- [UTF-8 Support in OLE DB Driver for SQL Server](#)
- [SQL Server Collation Name \(Transact-SQL\)](#)
- [Windows Collation Name \(Transact-SQL\)](#)
- [Introducing UTF-8 support for SQL Server](#) 
- [COLLATE \(Transact-SQL\)](#)
- [Collation Precedence](#)

See also

- [Contained Database Collations](#)
- [Choose a Language When Creating a Full-Text Index](#)
- [sys.fn_helpcollations \(Transact-SQL\)](#)
- [Single-Byte and Multibyte Character Sets](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)