



# All You Wanted to Know About **Collations**

Erland Sommarskog  
Data Platform MVP





**Erland Sommarskog**

**Independent consultant based in Stockholm**

**SQL Server MVP since 2001**

**esquel@sommarskog.se**

**<http://www.sommarskog.se>**

Slides and scripts are available on

<http://www.sommarskog.se/present>



# Agenda

- Collation basics.
- Unicode and code pages.
- The anatomy of a collation name.
- UTF-8 collations.
- LIKE ranges.
- Collation conflicts.
- Time permitting: Two performance cases.



# What Is a Collation?

A collation is a set of rules for how to handle string data depending on human language. This includes:

- Comparison. Is 'I' = 'i'? 'V' = 'W'? 'Ö' > 'Z'? '+' > '-'?
- Grouping and sorting.
- Result of upper/lower.
- LIKE ranges, for instance `col LIKE '[A-Z]%'`.
- Which code points in Unicode that are defined.
- The code page for varchar.

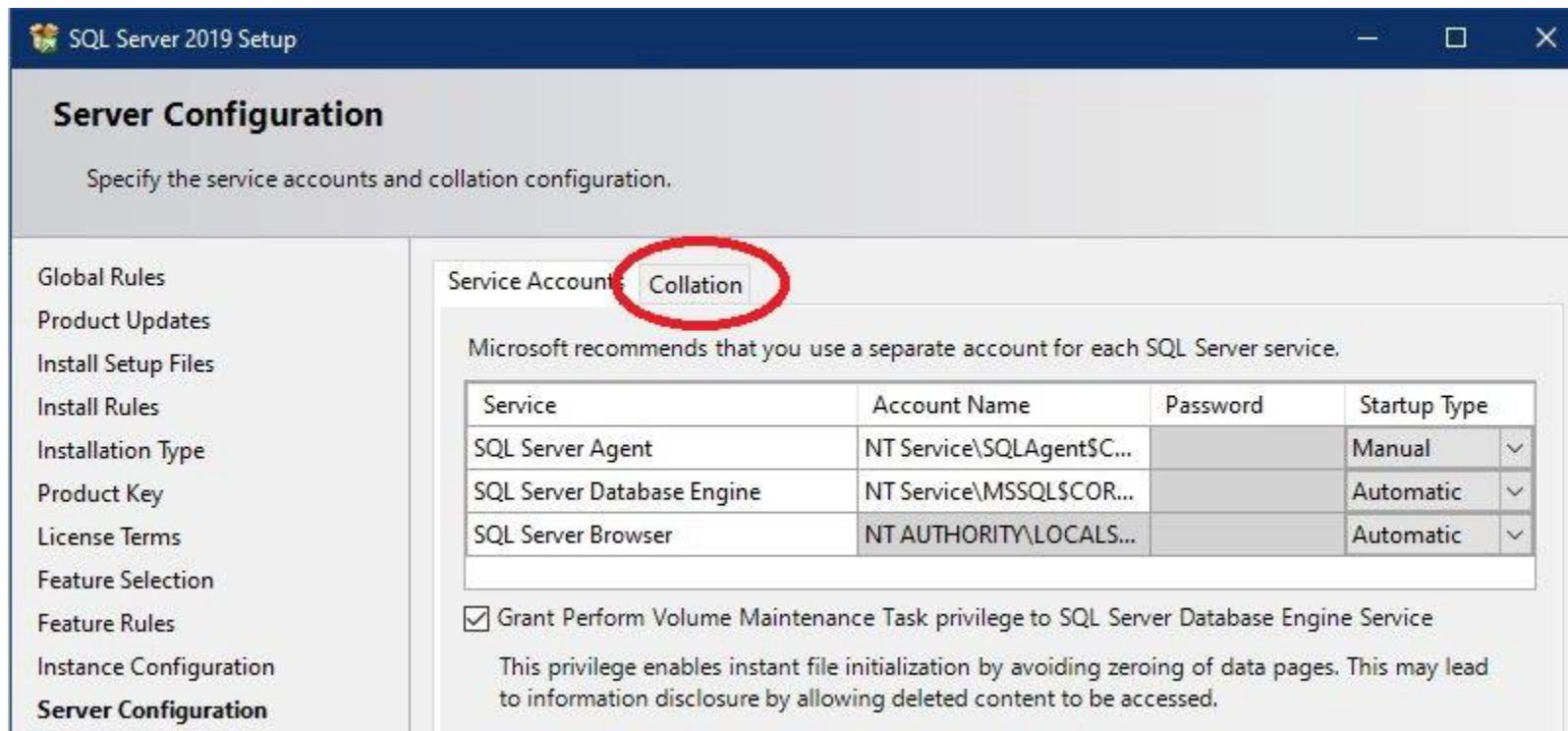
# Four Levels to Set Collations

- **Server collation – set when you install SQL Server.**
  - Collation for the system databases.
  - Default collation for string columns in temp tables.
  - The default for:
- **Database collation – may override server collation.**
  - Defines the collation for string variables and literals.
  - Sets the default for collation in user tables and table variables.
- **Column collation – may override database setting.**
- **Expression level – to cast collation when needed.**



# Collations and Setup

- In Setup wizard, Collation is tucked away on a secondary tab:



# Collations and Setup

- Always check what is on that secondary tab – it may not be what you expect.
- The default in the Setup Wizard is taken from the Windows system locale – which may differ from your personal regional settings.
- For some system locales, the default collation is not appropriate.
- Changing a collation after the fact is painful!
  - [This article](#) on my web site can help you.



# Getting Collation Information

**Server collation:**

```
SELECT serverproperty('Collation')
```

**Database collation:**

```
SELECT databasepropertyex('MyDB', 'Collation')
```

Or look in sys.databases.

**Column collation:**

```
EXEC sp_help tablename
```

Or look in sys.columns.





# The Unicode Character Set

- Windows is based on *Unicode*, a character set which supports all human languages.
- 1.1 million possible code points (21 bits). Currently, 155 000 are defined.
- New versions of Unicode define more code points.
  - A collation is based on a certain Unicode version.
- All living languages can be expressed with the “base plane” – code points 0 to 65535 (16 bits).
- Code points 32-126 = ASCII. 160-255 = ISO Latin-1.



# Unicode Encodings

- Sample code points: **A** = U+0041, **ö** = U+00F6, **α** = U+03B1, **中** = U+4E2D, **♠** = U+1F0A1.
- Unicode can be encoded in several ways. Most commonly used are:
  - UTF-8, one to four bytes per code point.
  - UTF-16, two bytes per code point for the base plane, four bytes per code point for the rest.
- Windows is based on UTF-16.
- In SQL Server, nvarchar is UTF-16.



# Legacy Code Pages

- Windows supports legacy charsets, called *code pages*.
- They all(?) have ASCII in the range 32-126 and use the range 127-255 to support one or more languages.
- Some examples:
  - CP850 (West Eur, proprietary from MS-DOS days).
  - CP932 (Japanese).
  - CP1250 (East Eur, ISO-8859-2, a.k.a. Latin-2).
  - CP1252 (West Eur, ISO-8859-1, a.k.a. Latin-1).
  - CP65001 (UTF8).
- These code pages are the foundation for varchar.
  - Varchar is not always 8-bit!



# All the Collations

- **In SQL 2019 and later, there are 5508 collations.**
  - In Azure SQL Database the number is 5540.
- **5431 Windows collations.**
  - Based on Windows system locales.
  - All operations are carried out in UTF-16 also for varchar.
- **77 SQL Collations, legacy from SQL 7 and earlier.**
  - For varchar, based on a specific code page with its own library.
  - For nvarchar they are like a Windows collation.
  - While legacy, the most commonly used collation is SQL\_Latin1\_General\_CP1\_CI\_AS, being the default in many contexts.



# The Anatomy of a Collation Name

- **Thai\_100\_CI\_AI\_KS\_WS\_SC\_UTF8** – Means what?
- **Thai** – Collation family.
- **100** – Version number.
- **CI/AI** – Case and accent (in)sensitivity.
- **KS/WS/VSS** – Japanese/East Asian.
- **BIN** and **BIN2** – Binary.
- **SC** – Surrogate compatible.
- **UTF8** support.



# Collation Families

- **Example of collation families:**
  - Thai, Polish, Finnish\_Swedish, Traditional\_Spanish, Modern\_Spanish, German\_PhoneBook, Latin1\_General.

**A collation family determines:**

- **Basic sorting and comparison rules.**
  - Further refined by CI/CS, AI/AS etc.
- **Rules for lower/upper.**
- **The code page for varchar. (Not UTF8 collations.)**



# The Version Number

- Can be none (=80), 90, 100, 140 or 160.
  - 80: The original collations in SQL 2000.
  - 90: Support for a few more languages in SQL 2005.
  - 100: New versions of the original and support for more languages in SQL 2008.
  - 140: Only Japanese collations.
  - 160: Two Chinese collation families in Azure SQL only.
- The version number is tied to a version of Unicode, and thus determines which code points that are defined.

[03\\_versionnumbers.sql](#)



# Case and Accent Sensitivity

- **CI** – Case insensitive. 'insert' = 'INSERT'.
- **CS** – Case sensitive. 'insert' <> 'INSERT'.
- **AI** – Accent insensitive. 'resume' = 'résumé'.
- **AS** – Accent sensitive. 'resume' <> 'résumé'.
- **CI\_AS** is the most commonly used combination, but **CI\_AI**, **CS\_AS** and **CS\_AI** are all possible.



# Collations and Metadata

- Due to legacy, collations also control metadata.
- The server collation controls:
  - Names of server-level objects (databases, logins etc).
  - Names of temp tables and column names in temp tables and table variables.
  - *Names* of variables. (Values follow the database collation.)
- The database collation controls:
  - Names of database-level objects (tables, columns, users etc).
  - Service-Broker objects have a binary collation.



# Japanese Matters – KS/WS/VSS

- **KS:** Japanese has two syllable scripts, Hiragana and Katakana.
  - Is なかめぐろ = ナカメグロ? (“Naka-Meguro”)
  - Yes, if no KS. No if KS (Kana-sensitive).
- **WS:** To match East-Asian and Latin characters in size, there are fullwidth and halfwidth forms.
  - Is **Paris** = P a r i s?
  - Is カメクロ = ナカメグロ?
  - Yes, if no WS. No if WS (Width-sensitive.)
- **VSS:** Variation-selector sensitive. I pass. :-)
  - Only in Japanese collations with version number = 140.



# Binary Collations

- Binary collations sort and compare by code point.
- They are case-, accent-, kana-, and everything else-sensitive.
- Not always user-friendly – but they are fast.
- There are two of them `_BIN` and `_BIN2`. [06 binarycollations.sql](#)
- `BIN2` are “normal”, sort all by code point.
- `BIN` are legacy. Swap the first byte only, and sort remaining by raw binary.



# SC – Supplementary Characters or Surrogate Compatible

- In UTF-16, characters beyond the base plane are encoded as so-called surrogate pairs.
  - High word is in the range U+D800 to U+DBFF.
  - Low word is in the range U+DC00 to U+DFFF.
- The original version-80, 90 and 100 collations only support the Unicode base plane, 0-65535.
- The \_SC collations support UTF-16 in full.
- All version-90 and later have an \_SC version.
  - Binary collations are not surrogate aware.

[07\\_surrogates.sql](#)





# UTF-8 Collations

- Introduced in SQL 2019. Suffix is `_UTF8`.
- All SC collations have a UTF-8 version.
  - Thus, there are no version-80 collations for UTF8.
- In a UTF8 collation the code page for varchar is always 65001 = UTF-8.
- UTF-8 collations do not support the text and ntext data types. (True for all SC collations.)
- There is one binary UTF-8 collation, `Latin1_General_100_BIN2_UTF8`.



# Saving Space with UTF-8?

- **The range 0-127 takes up one byte.**
  - ASCII. Up to 50 % space saving for English.
- **The range 128-2047 is encoded with two bytes.**
  - Latin, Cyrillic, Greek, Arabic, and a few more scripts.
  - Still 5-10 % space saving for non-Latin scripts.
- **The range 2048-65535 need three bytes.**
  - Languages of India, Chinese, Japanese, Thai etc.
  - Up to 50 % more space.
- **Beyond the base plane, four bytes.**
  - No difference to UTF-16.

# UTF-8 and String Length

- `varchar(30)` means 30 *bytes*.
- To permit for 30 characters, you may need `varchar(60)` – and still take your chances that you will not need to store Chinese or emojis.
- Then again, a 10-byte value in a `varchar(60)` does not take up more space than in a `varchar(30)`.



# UTF-8 or UTF-16 (nvarchar)?

- Personally, for a new application I would still go with nvarchar for international support.
  - More predictable.
  - Better performance.
  - Space can be saved with row compression as well.
  - But it certainly is a matter of preference.
- For an existing app with varchar that faces international needs, moving to a UTF-8 collation can be less painful than switching to nvarchar.
  - But it may still be a lot of work. [My article](#) has a few tips.





# LIKE Ranges

- Some people try `WHERE word LIKE '[A-Z]%'` to find words starting in uppercase.
  - They have regular expressions from Unix, text editors etc in mind.
- But **LIKE** ranges are also subject to collations as well...
- Forcing a binary collation helps in this case.

[09 LIKE-ranges.sql](#)



# Collation Conflicts

- When columns with different collations meet you get a conflict.

[11\\_collation\\_conflict.sql](#)

- Resolve with COLLATE:

```
JOIN a ON a.col COLLATE Latin1_General_CI_AS = b.col
```

- This casts the collation for *both* columns.
- Beware that casting the collation kills any index.
- Best Practice: Use COLLATE DATABASE\_DEFAULT in temp tables.
- Columns always win over variables/constants.



# Collations and Performance

- Does the collation affect performance?
- Yes – but effect is moderate, and evened out over the entire workload, it may not be noticeable.
- You select the collation from business needs, not from performance.
- But there are two situations where the collation has an enormous impact.



# The Data-Type Mismatch Accident

- Consider this: `WHERE indexedvarchar = @nvarchar`.
  - A programming mistake, but it can happen easily.
- The varchar column is converted to nvarchar.
- Windows collation: varchar is an ordered subset of nvarchar.
- => Index Seek with some extra operators. Expect an overhead of 2-3x – which for a quick lookup is not much.





# SQL Collations Are Different

[12 SQL-collation.sql](#)

- **SQL collation: varchar sorts and compares differently from nvarchar.**
- **Thus, in case of a data-type mismatch, index is dead and there must be a scan.**
- **Performance disaster. Can be a million times slower. Or more.**



# WHERE stringcol LIKE '%abc%'

- Not only must there be a scan of index/table, but SQL Server must also scan each string.
- With `nvarchar` or a Windows collation, the complex Unicode rules are applied to every character – that adds up.
- For *varchar*, SQL collations are 7-10 times faster than Windows collations.
  - Because there are only 255 characters to consider.



# Binary Collations to the Rescue?

- Binary Windows collations are even faster – but are case-, accent- etc sensitive.
- However, try this:

```
upper(col) COLLATE Latin1_General_BIN2 LIKE upper(@str)
```

- Works for both varchar and nvarchar.
- Not entirely foolproof – but may be good enough.

# The Final Slide

Erland Sommarskog,  
[esquel@sommarskog.se](mailto:esquel@sommarskog.se)

Slides and scripts on  
<http://www.sommarskog.se/present>

Clean-up script: [14 cleanup.sql](#)

