

All You Wanted to Know About *Collations*

Erland Sommarskog

Data Platform MVP



Accelerating Data-Driven Success





Erland Sommarskog

Independent consultant based in Stockholm

SQL Server MVP since 2001

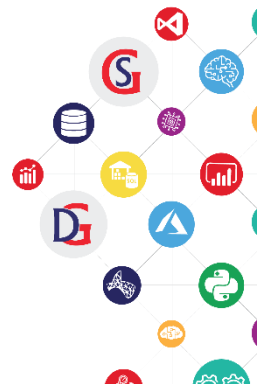
<http://www.sommarskog.se>

esquel@sommarskog.se

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

Agenda

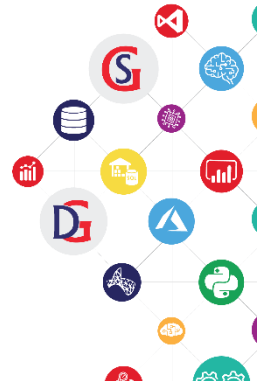
- **Collation Basics.**
 - What is a collation?
 - Where to define collations in SQL Server.
- **A history of character sets and code pages.**
- **All the collations.**
 - Windows collations.
 - The anatomy of a collation name.
 - UTF-8 collations.
 - SQL collations.
- **Collation pain points:**
 - Metadata.
 - Collation conflicts.
 - Changing database collation.
- **Some tricks with collations.**
- **Performance.**



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.
- Sorting.
- Result of upper/lower.
- LIKE ranges, for instance `col LIKE '[A-Z]%'`.
- We will add two things later.



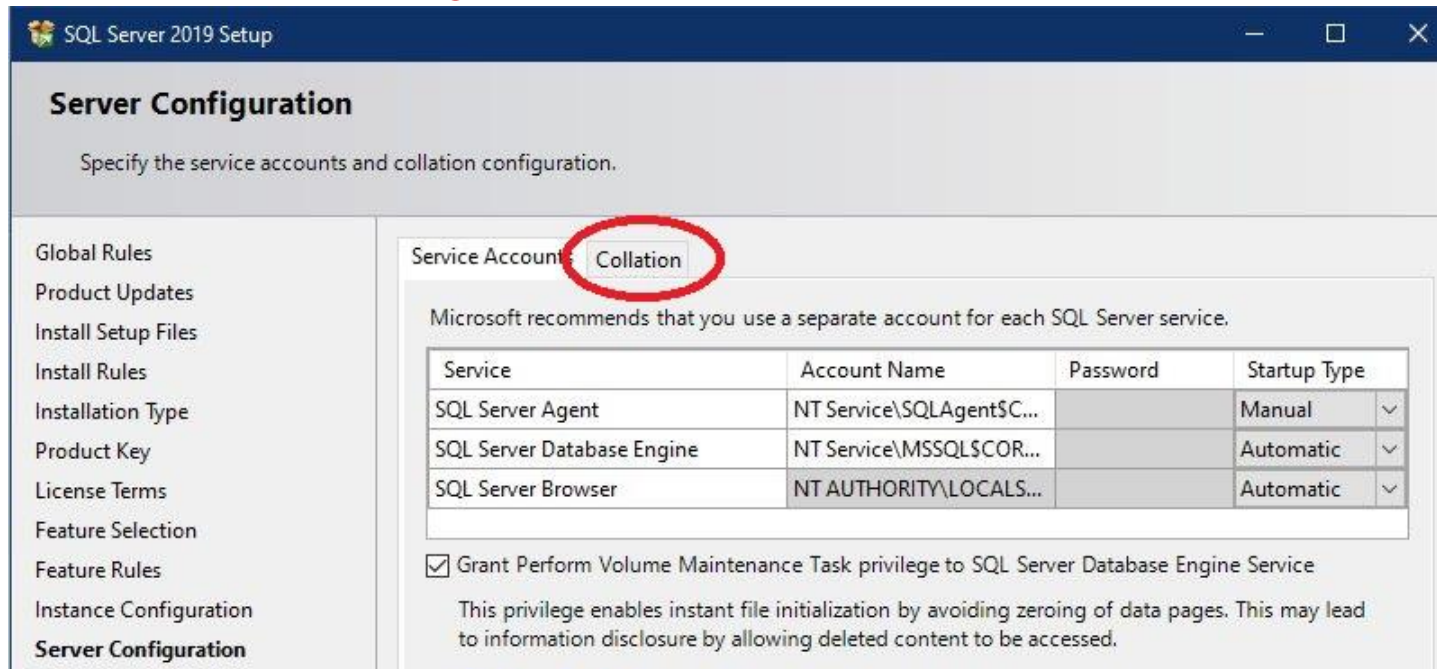
Four Levels to Set Collation

- **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** – cast collation on the fly.



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 is taken from the Windows system locale – not your personal regional settings.
- The default collation is far from always the best choice – or even relevant for the system locale.
 - See list [here](#). (Scroll to *Server-level collations*.)
- Changing a collation after the fact is painful!



Getting Collation Information

Server collation:

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

Database collation:

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

Or look in `sys.databases`.

Column collation:

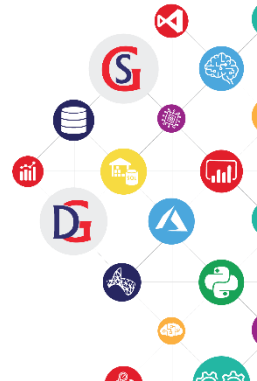
```
EXEC sp_help tablename
```

Or look in `sys.columns`.



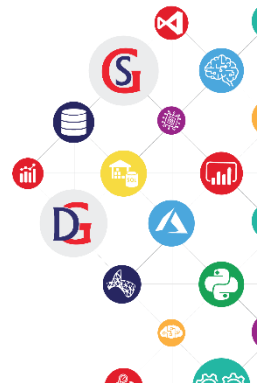
A Brief History of Character Sets and Code Pages

- In the beginning there was ASCII...
- A 7-bit character set that encodes 95 characters with codes from 32 to 126.
- Example: A = 65 / 0x41, a = 97 / 0x61 etc.
- Designed with English in mind.
- There were variations for ASCII for other languages, but they are dead now.



Advent of 8-bit Character Sets

- In the 1980s, several vendors designed 8-bit charsets to support groups of selected languages.
- They all used ASCII for the range 32-126 and added more characters in the range 128/160-255.
- They were proprietary and incompatible.
- The standard ISO 8859 emerged, a series of sets.
 - 8859-1, Latin-1, Western European languages.
 - 8859-2, Latin-2, Eastern European languages.
 - There were also sets for Arabic, Cyrillic, Greek and Hebrew.



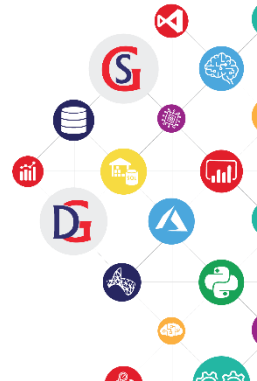
8-bit Sets in the Microsoft World

- MS-DOS had its proprietary character sets, known as *code pages*.
 - CP437 (The original), CP850 (West Eur), CP852 (East Eur).
- Windows adapted ISO 8859, still calling them code pages.
 - CP1250 (East Eur), CP1251 (Cyrillic), CP1252 (West Eur).
 - Filled in the gap 128-159 with extras.



Enter Unicode

- A single character set that encodes *all* languages.
- 1.1 million possible code points (21 bits), 143 000 are currently defined.
- New versions of Unicode define more code points.
- 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.
- Sample code points: A = U+0041, Ł = U+0141, α = U+03B1, 中 = U+4E2D, ♠ = U+1F0A1.



Unicode Encodings

- **Unicode can be encoded in several ways:**
 - UTF-32, four bytes per code point, rarely used.
 - UTF-16, two bytes per code point for the base plane, four bytes per code point for the rest.
 - UTF-8, one to four bytes per code point.
- **Windows is based on UTF-16.**
- **UTF-8 is just another code page, CP65001.**



String Data Types in SQL Server

- **n(var)char**
 - Unicode, stored as UTF-16.
 - String literals preceded by N, for instance N'My Text'.
- **(var)char**
 - String literals without N, for instance 'My Text'.
 - A collation is tied to a code page.
 - And thus, the collation defines the available characters.
 - Unavailable characters are replaced by fallbacks.
 - Not all collations support (var)char!
 - It is not necessarily 8-bit!

[01_codepages.sql](#)



All the Collations

- **In SQL 2019 there are in total 5508 collations.**
 - **Plus some 180 deprecated collations.**

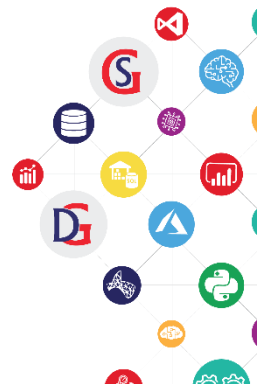
They fall into two main groups:

- **77 SQL collations (Legacy).**
 - The name starts with SQL_.
- **5431 Windows collations.**
 - Based on Windows system locales.
 - All operations are carried out in UTF-16, also for varchar.



Anatomy of a Collation Name

- **Polish_100_CI_AI_KS_WS_SC_UTF8** – Means what?
- **Polish** – Collation family.
- **100** – Version number.
- **CI/AI** – Case and accent (in)sensitivity.
- **KS/WS/VSS** – Japanese/East Asian.
- **SC** – Supplementary characters.
- **BIN** and **BIN2** – Binary.
- **UTF8** support.



Collation Families

- **Example of collation families:**
 - Polish, Assamese, Traditional_Spanish, Modern_Spanish, Finnish_Swedish, Latin1_General, Indic_General.

A collation family determines:

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

[02_collationfamilies.sql](#)



Collation Families Without varchar

- **Assamese**
- **Bengali**
- **Divehi**
- **Indic_General**
- **Khmer**
- **Lao**
- **Maltese**
- **Maori**
- **Nepali**
- **Pashto**
- **Syriac**
- **Tibetan**

Recall: does not apply to UTF8 collations.



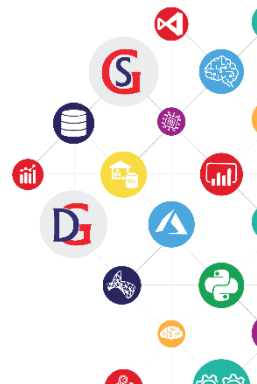
The Version Number

- **Can be none (=80), 90, 100 or 140.**
 - 80: The original collations in SQL 2000.
 - 90: More collations to support for a few more languages.
 - 100: New versions of the original collations and support for even more languages.
 - 140: Only Japanese collations.
- **The version number relates to a version of Unicode.**
- **Determines which code points that are defined.**

03_versionnumbers.sql

Undefined Code Points

- **When a code point is undefined, unexpected things may occur:**
 - `charindex()`, `LIKE` etc do not find the character.
 - `replace()` does not replace the character.
 - The character is ignored in comparisons.
 - `upper()/lower()` have no effect.
- **Depending on the language you work with, you may need a version-100 collation.**
- **Versions of Unicode may give you some idea.**



Case and Accent Sensitivity

- **CI – Case-insensitive.** 'insert' = 'INSERT'.
- **CS – Case-sensitive.** 'insert' <> 'INSERT'.
- **AI – Accent-insensitive.** 'resume' = 'résumé'.
- **AS – Accent-sensitive.** 'resume' <> 'résumé'.

[04 CICS AIAS.sql](#)

- **Watch out for Turkish!**
- **Accent-insensitive comes with surprises.**



Sorting and Case/Accent Sensitivity

- When sorting in a CS collation, case carries a secondary weight.
 - That is, case only affects sorting when nothing else differs.
 - Lowercase sorts before uppercase.
 - Hyphens in words only also have a secondary weight.
 - Not true for other punctuation.
 - Analogously, accents carry secondary weight in AS collations.
- 
- A set of small, semi-transparent navigation icons located in the bottom right corner of the slide. These icons include symbols for navigating between slides, such as arrows, a search icon, and other standard presentation controls.



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 ナカメグロ = 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.sc](#)
- `BIN2` are “normal”, sort all by code point.
- `BIN` are legacy. Swap the first byte only, and sort remaining by raw binary.

06 binarycollations.sql



SC – Supplementary Characters

- The original version-80, -90 and -100 collations only support the Unicode base plane, 0-65535.
- SC collations also support supplementary characters, those beyond the base plane.
- In UTF-16, they 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.
- All version-90 and -100 collations have an _SC version.
 - Version-140 collations are always surrogate compatible.
 - Binary collations are not surrogate-aware.

07 surrogates.sql



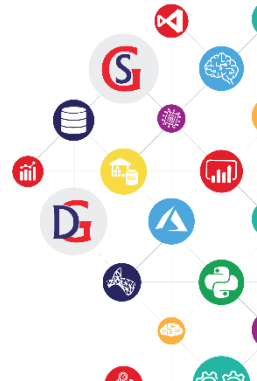
When are Supplementary Characters Needed?

- I looked in Wikipedia to find examples.
- Lots of ancient scripts.
- A full plane of “extra ideographic characters”.
- Various symbols, a selected few:
 - Playing cards.
 - Transportation and Map Symbols, for instance: 
 - Musical symbols.
- ...and emojis/emoticons.



UTF-8 Collations

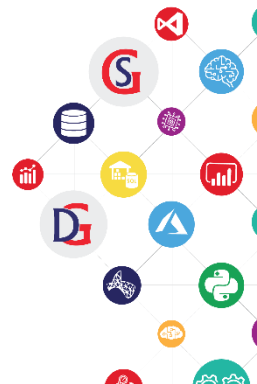
- Introduced in SQL 2019. Suffix is `_UTF8`.
- All SC collations have a UTF-8 version.
 - Thus, there are no version-80 collations for UTF8.
- In a UTF8 collation the code page for varchar is always 65001 = UTF-8.
 - So no restriction for Indic_General etc!
- UTF-8 collations do not support the text 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?

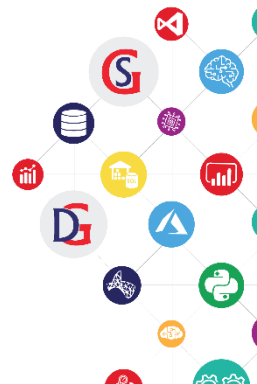
- **The range 0-127 takes up one byte.**
 - ASCII. Up to 50 % space saving for English.
- **The range 128-2047 is encoded with two bytes.**
 - Latin, Cyrillic, Greek, Arabic, and a few more scripts.
 - Still 5-10 % space saving for non-Latin scripts.
- **The range 2048-65535 needs three bytes.**
 - Languages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

08 UTF8.sql



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.**
 - Presumably in short columns for names etc, you do not have to account for emojis.
 - **Somewhat better performance.**
 - We will return to this later.
 - **Space can be saved with row compression as well.**
 - In a test, I got exactly the same size with UTF-16 and UTF-8 when row-compressed.
 - **But it certainly is a matter of preference.**



UTF-8 or UTF-16, Existing Apps

- Say that you have an existing application using varchar with a legacy code page that faces international needs.
- Switch to nvarchar:
 - Lot of code go through – just think of all variable declarations and temp tables.
- Change to UTF-8 collation:
 - If you can keep column lengths, you have a free ride!
 - If not, there is still a bit work. (But still less than nvarchar).



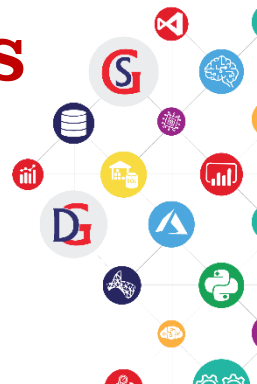
By the Way...

- For new applications always use Unicode (nvarchar or varchar with UTF-8) for names, free-text etc.
 - Do not use varchar with a legacy code page for such columns.
- Don't think “we will never be international”. We live in an global and rapidly changing world.
- Changing to Unicode later can be very expensive.
- Codes, status columns, transaction ids are often ASCII only – here a legacy code page is OK.



SQL Collations

- Legacy collations, from SQL 7 and earlier.
- Up to SQL 7, SQL Server only supported one collation (sortorder) in an instance.
 - And up to SQL 6.5 there was only varchar.
- For nvarchar, an SQL collation is like a version-80 Windows collation.
- For varchar, based on a specific code page with its own library and rules.



SQL_Latin1_General_CP1_CI_AS

- While legacy, this is surely the most commonly used collation.
- It's the default for:
 - System locale is English (United States).
 - All other system locales default to a Windows collation!
 - On Linux and Docker.
 - Everything you do in Azure.
 - And for an Azure SQL VM, you don't even get the choice.

09 SQL-collation.sql



SQL Collations and (var)char

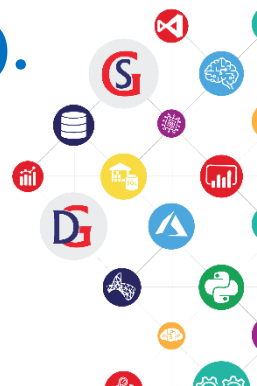
- In SQL_Latin1_General_CP1_CI_AS, the letters Œ, Š and Ž are case-sensitive and sorted like symbols.
 - Presumably of legacy reasons.
 - Some SQL collations handle them correctly.
- Some punctuation characters sort differently.
- A case-sensitive SQL collation sorts uppercase before lowercase.
 - No special handling of the hyphen.



Collations and Metadata

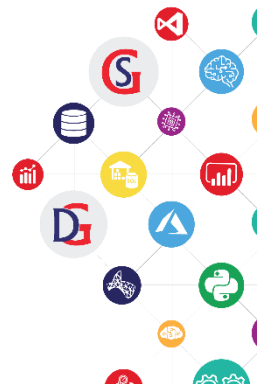
- Due to legacy, collations also control metadata.
- The server collation controls:
 - Names of server-level objects (databases, logins etc).
 - Names of temp tables and column names in temp tables and table variables.
 - *Names* of variables. (Values follow the database collation.)
- The database collation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

[10_metadata.sql](#)



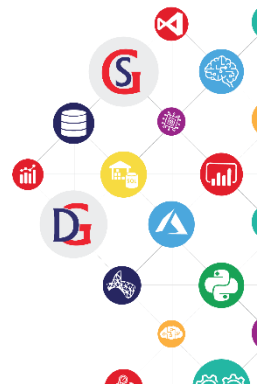
Tips for International Software

- **Develop with a CS collation, both on server-level and database-level.**
 - Or forget about the Turkish market.
- **Corollary: Use only lowercase for identifiers.**
- **Avoid very short names like #v or @w to avoid accent-insensitive surprises.**



Fixed Metadata Collation?

- With a contained database, you get a fixed metadata collation, Latin1_General_100_CI_AS_KS_WS_SC.
- Alas, there are restrictions with contained databases, so it is difficult to recommend them.
- I think this should be an option for regular databases. [Vote here!](#)



Collation Conflicts

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

[11_collation_conflict.sql](#)

- Resolve with COLLATE:

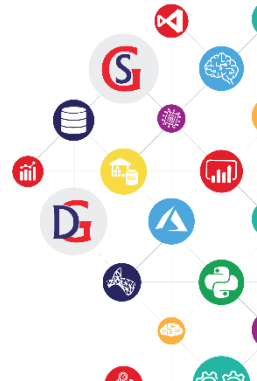
```
JOIN a ON a.col COLLATE Latin1_General_CI_AS = b.col
```

- 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.



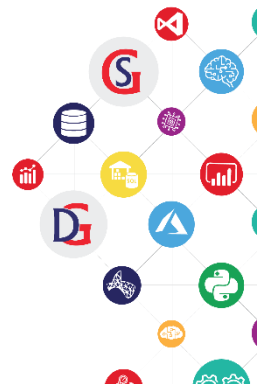
Changing Database Collation

- **This is hard!**
- **ALTER DATABASE db COLLATE**
 - Changes the collation of the system tables only.
 - Gives up if there is a single CHECK constraint.
- **For user tables: ALTER TABLE ALTER COLUMN**
 - Must repeat full column definition.
 - Indexes must be dropped and restored.
 - Same goes for foreign-key and CHECK constraints.
 - Constraints and unique indexes may fail when restored.



Tips for Changing DB Collation

- Take as many shortcuts as you can.
- Sometimes it may be better to create a new database and copy data over.
- For a script: See Hugo Kornelis's [blog post](#). Newest version on [Github](#) by Bas Roovers.
 - You may have to refine further.
- There are blog posts about startup option -q for SQL Server.
 - Undocumented and unsupported, do not use!



Useful Tricks with Collations

- Removing “accents”.
- Finding words that starts with uppercase.
- Removing invisible characters.
- Casting to a non-Latin collation and then to varchar will replace accented characters with fallbacks.
- LIKE ranges depends on the collation, so you often need a binary collation.
- Some control characters are undefined in Windows collations – cast to a binary collation to resolve.

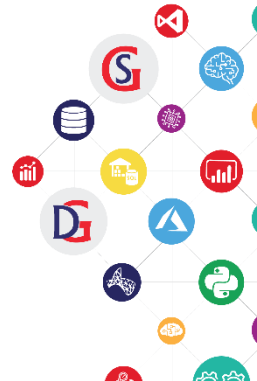
12 useful tricks.sql



Collations and Performance

[13_performance.sql](#)

- **Lookup with nvarchar value on an indexed varchar column.**
 - A programming mistake, but can easily happen.
- **Windows collation: 2-5 times slower, but index still sought.**
 - Possible because varchar is an ordered subset of nvarchar.
- **SQL collation: index is scanned – performance disaster.**
 - Because rules for varchar are different from nvarchar.



Performance with LIKE '%abc%'

- **For varchar, SQL collation is 7-10 times faster than a Windows collation.**
 - Windows collations and nvarchar are expensive, because the complex Unicode rules are applied to every character.
- **Binary Windows collations are even faster – but are case-, accent- etc insensitive.**
- **Tip: Lower/upper() can make a binary search case-insensitive and good enough – and still as fast as an SQL collation.**



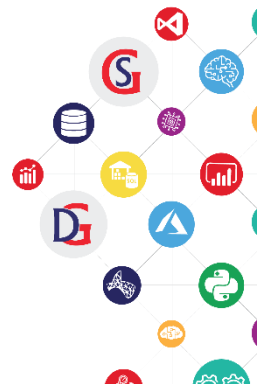
Testing Performance

- I tested all 5508 collations.
- Test ran for nine days, testing five different operations, for both varchar and nvarchar.
- Platform was SQL 2019.
 - I also ran similar tests in 2008 on SQL 2008.
- Some selected data follows.
- Caveat: The effect on an real-world workload is likely be far less than the numbers indicate.



Data from Performance Tests

- **CI_AI** about 3-6 % faster than **CS_AS**.
- **100-collations** 5-30 % slower than **80-collations**.
 - Because character tables are larger.
- For Windows collations (not UTF-8), **varchar** is 0-50 % slower than **nvarchar** depending on operation.
 - Because of conversion to UTF-16 and back.
- For SQL collations, **varchar** is faster, if not always as extreme as with **LIKE**.
- Some languages are slower than others.



Performance of UTF8

- Comparing UTF-8 with varchar to nvarchar in version-80 collation:
 - LIKE '%abc%': 12 % slower.
 - Point lookup: 22 % slower.
 - GROUP BY (hash match): 27 % slower.
 - Loop join: 45 % slower.
 - ORDER BY: 110 % slower.



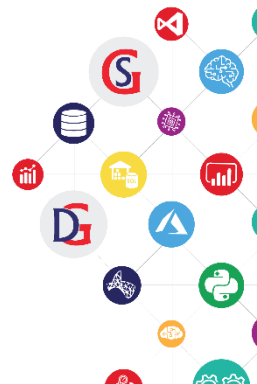
Binary Collations

- **Binary vs. normal collations:**
 - GROUP BY (hash match): 15-25 % time reduction.
 - Point lookup: 7-27 % time reduction.
 - LOOP JOIN: 15-36 % time reduction.
 - ORDER BY: 40-70 % time reduction.
 - In a real-world test case, I gained 10-30 %.
- **BIN vs. BIN2 – generally BIN2 is marginally faster.**
 - Only for ORDER BY operation it was more than 10 %.

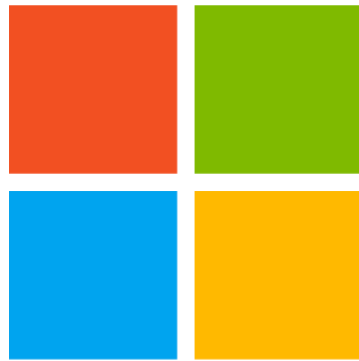


Caveats

- Take these performance data with a big grain of salt.
- Very synthetic tests that test specific operations.
- They do not take in account the effect of the size etc.
- A system has more operations than string operations.
- If you really want to know – test with your workload.



Special Thanks To

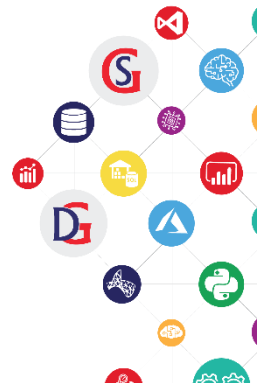


Microsoft

for supporting

DataPlatformGeeks & SQLServerGeeks

Community Initiatives



Thank You

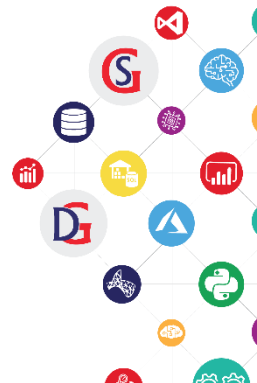
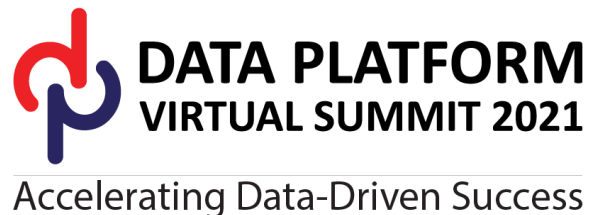
Three Ways to Win Prizes

Post your selfie with hash tag **#DPS2021**

Give Session & Conference Feedback

Visit our Sponsors & Exhibitors

Follow us on Twitter **@TheDataGeeks @DataAISummit**



The Final Slide

Erland Sommarskog
esquel@sommarskog.se

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

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

