SQLintersection

Session: 11/19/2019, 2:15pm – 3pm

Choices, Choices…
Using Unicode in SQL Server and Azure SQL

Pedro Lopes
@SQLPedro

SQL intersection

Microsoft

Speaker: Pedro Lopes

- Program Manager @ Azure Data SQL Server team
- Relational Engine: Query processing; Performance
- Compatibility Certification (https://aka.ms/dbcompat)


/pedroazevedolopes


@SQLPedro

Reminder: Intersect with Speakers and Attendees

- **Tweet *tips and tricks* that you learn and follow tweets posted by your peers!**
  - Follow: #SQLintersection and/or #DEVintersection
- **Join us – Wednesday Evening – for SQLafterDark**
  - Doors open at 7:00 pm
  - Trivia game starts at 7:30 pm

    Winning team receives something fun!
  - Raffle at the end of the night

    Lots of great items to win including a seat in a five-day SQLskills Immersion Event!
  - The first round of drinks is sponsored by SentryOne and SQLskills

SQL intersection

SQLskills immerse yourself in sql server & SentryOne™

Overview

- **Why would you care about Unicode?**
- **What is Unicode?**
- **String encoding in SQL Server**
  - What's been there forever
  - What's new?
- **Which one do you choose?**
  - UTF-16
  - UTF-8
  - Perf and functional comparisons
- **Conversion Methods**

SQL intersection

Unicode

Why would you care?

Contoso Inc.

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

Think through the challenges…

Working with languages

Storing data in multiple languages within one database is difficult to manage when using code pages

- A code page is a representation of a **character set**: a specific subset of language specific characters, out of the entire repertoire of known characters
- But a code page is simultaneously a **character encoding scheme**: a direct mapping of characters to bytes (octets)

Code pages are set by the OS and impact installed applications. For example:

- CP874: Thai
- CP1250: Central European and Eastern European
- CP1251: Cyrillic
- **CP1252: Western** (similar to DOS 850 Latin-1 and 825 Latin-2)
- CP1253: Greek
- CP1254: Turkish
- CP1255: Hebrew
- CP1256: Arabic
- …

Contoso Inc.

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

Later, the company expands to parts of Asia and Eastern Europe.

Now the database must store customer names and addresses from:

- Russia / Serbia
- China
- Thailand
- Korea
- Japan

New challenges for Contoso, Inc

- Find one code page for the database that can store all the required language-specific characters

- Need 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.
  - **Incorrect translations = Gibberish = data loss!**

- Requirement of providing global multilingual database applications and services.
  - Databases that support international clients should always use **Unicode** data types instead of non-Unicode types

What is Unicode?

What is Unicode? Defining terms

Unlike code pages, Unicode separates the character set from the character encoding

- Unicode can represent the entire repertoire of known characters (the Universal Character Set)

Unicode is a standard for mapping code points to characters

- **Code points** mark the position of a numerical value in a code space:
  - ASCII code space has 128 code points (0-127)
  - Entire UCS code space has 1,114,111 code points
- **Characters** are representations of code point values. You can have **visible** and **non-visible** characters

```
SELECT ASCII('P') AS [ASCII],
       CHAR(80) AS [CHARACTER]
```

ASCII	CHARACTER
80	P

```
SELECT UNICODE('東') AS [UNICODE],
       NCHAR(26481) AS [CHARACTER]
```

UNICODE	CHARACTER
26481	東

What is Unicode? Common encoding schemes

UCS-2

- From UCS's 1.1 million characters, UCS-2 can represent the first 65,536 characters: the Basic Multilingual Plane (BMP)
- Uses 2 bytes per character (1 byte-pair)

UTF-8

- Can represent all of UCS's repertoire (BMP + Supplementary Characters)
- Variable-width encoding (1 to 4 bytes per character)
- Most popular. Mandatory for some systems in the EU and Japan for example.

UTF-16

- Evolution from UCS-2
- Can represent all of UCS's repertoire
- Variable-width encoding (1 or 2 byte-pairs per character)
- Used internally by Windows, Java and JavaScript

Why use UTF-8?

Maximum compatibility!

- The Web Hypertext Application Technology Working Group (**WHATWG**) considers UTF-8 a mandatory text encoding scheme.
- The World Wide Web Consortium (**W3C**) recommends UTF-8 as the default encoding in XML and HTML. More than 93%* of all web pages are encoded in UTF-8
- The Internet Engineering Task Force (**IETF**) requires all Internet protocols to identify the encoding used for character data, and the supported character encodings must include UTF-8
- The Internet Mail Consortium (**IMC**) recommends that all e-mail programs be able to display and create mail using UTF-8

* "Usage Survey of Character Encodings broken down by Ranking". *w3techs.com*. Retrieved 2019-04-01.

String encoding in SQL Server

SQL intersection

How does SQL Server encode strings?

Encodings are set in SQL Server through a combination of using the right **data type** with the right **Collation**

SQL Server 2012

Data Type	Collation flag: _SC	Collation flag: All others
NCHAR/NVARCHAR	UTF-16	UCS-2
CHAR/VARCHAR	Code page	Code page

Supplementary Character collation examples:

- Latin1_General_100_CI_AS_SC
- Japanese_Bushu_Kakusu_100_CI_AS_SC / Japanese_XJIS_140_CI_AS
- Cyrillic_General_100_CI_AS_SC

Note: Starting with SQL Server 2014, all _140 collations automatically support SC

How does SQL Server encode strings?

Encodings are set in SQL Server through a combination of using the right **data type** with the right **Collation**

SQL Server 2019
Azure SQL DB

Data Type	Collation flag		
	_SC	_UTF8	All others
NCHAR/NVARCHAR	UTF-16	UTF-16	UCS-2
CHAR/VARCHAR	Code page	UTF-8	Code page

UTF-8 collation examples:

- Latin1_General_100_CI_AS_SC_UTF8
- Japanese_Bushu_Kakusu_100_CI_AS_SC_UTF8 / Japanese_XJIS_140_CI_AS_UTF8
- Cyrillic_General_100_CI_AS_SC_UTF8

Note: All UTF-8 collations are SC-enabled

Demo

Implementing UTF-8

SQL intersection

So what do collations set?

Collations instruct SQL Server how to do three things:

Compare

- Binary
  - BIN compared the first character as WCHAR and then code-point comparison
  - BIN2 is pure code-point comparison
- Linguistic
  - Use rules set by the language, version, and flags set in the collation
  - CI; CS; AI; AS; KS; …

Upper/Lowercase

- Depends entirely on the language used in the collation name, and the version of the linguistic algorithm
  - **Albanian_100**_CI_AI
  - Apply equally to Binary and Linguistic collations

Encode

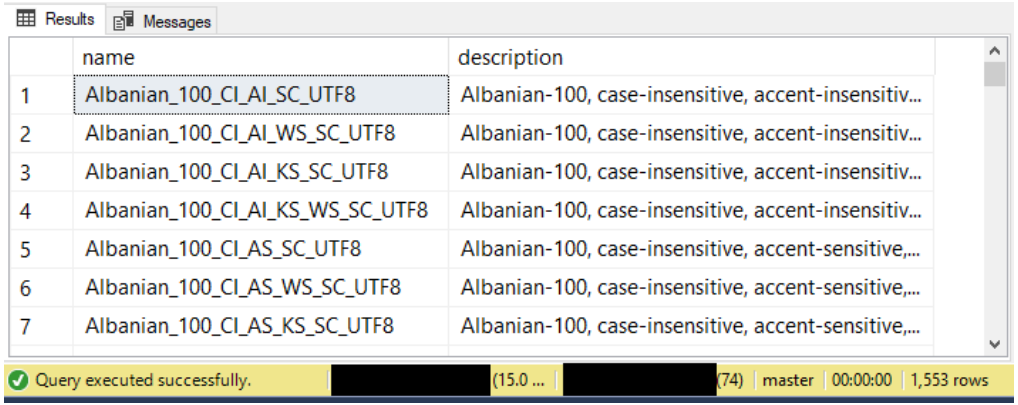
- Before UTF-8, CHAR/VARCHAR data encoding was entirely dependent on the language name used for the collation
  - **Albanian**_100_BIN or **Latin1_General**_CI_AI were internally mapped to code page 1252 (Latin)
  - **Japanese**_CI_AI was mapped to code page 932
- With UTF-8, data encoding has been decoupled from linguistic algorithms. For UTF-8 collations, data encoding is always UTF-8 (internally code page 65001)

So one part of encoding is the collation

UTF-8 is only available to Windows collations that support supplementary characters (_SC or 140), as introduced in SQL Server 2012

To find which collations support UTF-8, you can run the following:

```
SELECT name, description
FROM sys.fn_helpcollations()
WHERE Name like '%UTF8';
```



And how do data types relate?

Data types have always been encoded the same way, right?

- NCHAR/NVARCHAR uses 2 bytes per character
  - `NCHAR(20)` is 40 bytes and 20 characters
- CHAR/VARCHAR uses 1 byte per character
  - `CHAR(20)` is 20 bytes and 20 characters

And how do data types relate?

Data types have always been encoded the same way, right?

- NCHAR/NVARCHAR uses 2 bytes per character
  - `NCHAR(20)` is 40 bytes and 20 characters
- CHAR/VARCHAR uses 1 byte per character
  - `CHAR(20)` is 20 bytes and 20 characters

Only in **Latin** alphabets. The real data sizes are collation-dependant:

- NCHAR/NVARCHAR uses 2 or 4 bytes per character depending on the character set
  - `NCHAR(20)` is 20 byte-pairs (40 bytes), undetermined number of characters
- CHAR/VARCHAR uses whatever bytes per character, as determined by the character set in use
  - `CHAR(20)` is 20 bytes, undetermined number of characters

Storage differences between UTF-8 and UTF-16

Example "code pages"	Code Range (decimal)	Storage bytes with UTF-8	Storage bytes with UTF-16
ASCII (majority of Latin alphabets)	0 - 127	1	2
Extended Latin (including accents), Greek, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac, Tāna, N'Ko	128 – 2,047	2	2
Chinese, Japanese, Korean	2,048 – 65,535	3	2
Supplementary character range such as emojis ☺ , musical Clef, mathematical symbols	65,536 – 1,114,111	4	4

Note: *Storage bytes* refers to the encoded byte length, not the data-type on-disk storage size.

Demo

Debunking a western myth on string data types

SQL intersection

Performance and operational considerations

Which one do you choose?

Which one do you choose? Storage and perf

If your dataset uses primarily ASCII characters (<128 range, which represent majority of Latin alphabets), choose UTF-8

- Up to 50% storage savings when compared to UTF-16: NCHAR(10) requires 20 bytes for storage, whereas CHAR(10) requires 10 bytes for the same Unicode string
- UTF-8 uses 1 byte per character
- UTF-16 uses 2 bytes per character (1 byte-pair)
- When doing intensive read/write I/O on UTF-8, we measured an average **35% improvement** over UTF-16 using clustered tables with a non-clustered index on the string column, and an average **11% improvement** over UTF-16 using a heap

If your dataset is predominantly Extended Latin, but also Greek, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac, Tāna and N'Ko (all 128 to 2047 range), either is fine

- Both UTF-8 and UTF-16 use 2 bytes per character
- Performance measurements usually **very similar** between UTF-8 and UTF-16 in non-ASCII ranges

Which one do you choose? Storage and perf

If your dataset uses primarily Chinese, Japanese, or Korean characters (2048 to 65535 range), choose UTF-16

- UTF-8 uses 3 bytes per character
- UTF-16 uses 2 bytes per character
- We measured about **25% degradation** for intensive read I/O when using UTF-8 instead of UTF-16

In the supplementary character range such as emojis ☺ (65536 to 1114111), either is fine

- Both UTF-8 and UTF-16 use 4 bytes per character

Additional performance considerations

Compute-intensive operations such as SORTs/MERGE joins

- UTF-16 is generally better than UTF-8 for the same dataset
- This is because a few internal conversions happen during these operations

HASH joins/LIKE/Inequality comparisons

- Perform slightly better in UTF-8 for the same dataset

Other QP

- In some cases, NVARCHAR(MAX) holding up to 8,000 characters can be replaced with VARCHAR(8000) in order to store the value in-row
- A non-binary string value can be searched, filtered, and indexed more efficiently

Other considerations

UTF-8 can be set at the database level

- For existing databases, this does not affect existing objects, only new objects
- Existing objects must be converted (more ahead)
- UTF-8 and non-UTF-8 can co-exist in the same database and even same table: the COLLATE keyword can be used to set at column level

Always measure your character data length, not the # characters

- DATALENGHT measures encoded storage bytes
- LEN measures number of characters. Not new. Only useful if 1 character = 1 byte

Other considerations

UTF-8 encoding is not supported with:

- XML data type
- In-memory OLTP
- Always Encrypted (with Enclaves only)

T-SQL string literals (strings without N') are collated in the collation of the current database

- This means that inserting a Unicode string into a Unicode column can result in data issues, if the database was using a code page.
- This is not new behavior, regardless of being UTF-16 or UTF-8

Demo

Perf Comparisons

SQL intersection

UTF-16 to UTF-8

Conversion methods

How to convert to UTF-8?

Imagine your column data is encoded in UCS-2/UTF-16 or non-Unicode:

```
CREATE TABLE dbo.MyTable (MyString NVARCHAR(50) COLLATE
Latin1_General_100_CI_AI_SC);
```

```
CREATE TABLE dbo.MyTable (MyString VARCHAR(50) COLLATE
Latin1_General_100_CI_AI);
```

Convert in-place

```
ALTER TABLE dbo.MyTable
ALTER COLUMN MyString VARCHAR(50) COLLATE
Latin1_General_100_CI_AI_SC_UTF8
```

Pros	Cons
• Easy to implement	• Possibly blocking operation • May pose an issue for large tables and busy applications

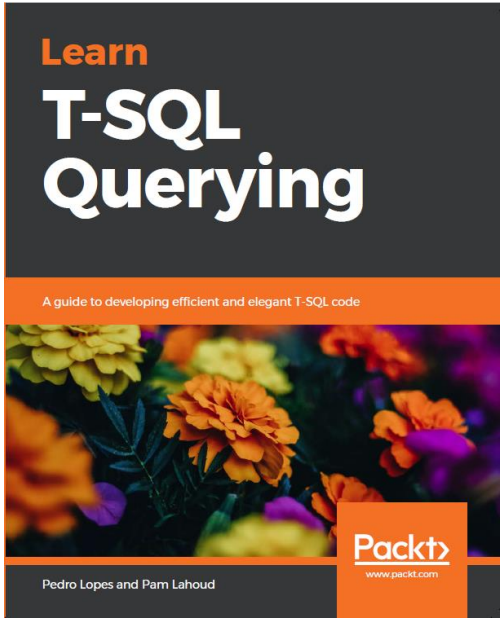
Copy and Replace

```
CREATE TABLE dbo.MyTable2 (VARCHAR(50) COLLATE
Latin1_General_100_CI_AI_SC_UTF8);
INSERT INTO dbo.MyTable2 SELECT * FROM dbo.MyTable;
DROP TABLE dbo.MyTable;
EXEC sp_rename 'dbo.MyTable2', 'dbo.MyTable’;
```

Pros	Cons
• Much faster than in-place	• Handling complex schemas with many dependencies (FKs, PKs, Triggers, DFs) • Tail of the table synch requires much more preparation

Learn more

Download and try SQL Server 2019	https://aka.ms/ss19
UTF-8 documentation	https://aka.ms/sqlutf8
Check out these great data-related demos	https://aka.ms/DataSamples https://aka.ms/IQPDemos https://aka.ms/SQL2019Notebooks
Continue learning with our new book	https://aka.ms/LearnTSQLQuerying https://aka.ms/LearnTSQLQuerying_errata
One shortcut to rule them all!	https://aka.ms/SQLShortcuts



Questions?

Microsoft

Don’t forget to complete an online evaluation!

Choices, Choices…
Using Unicode in SQL Server and Azure SQL

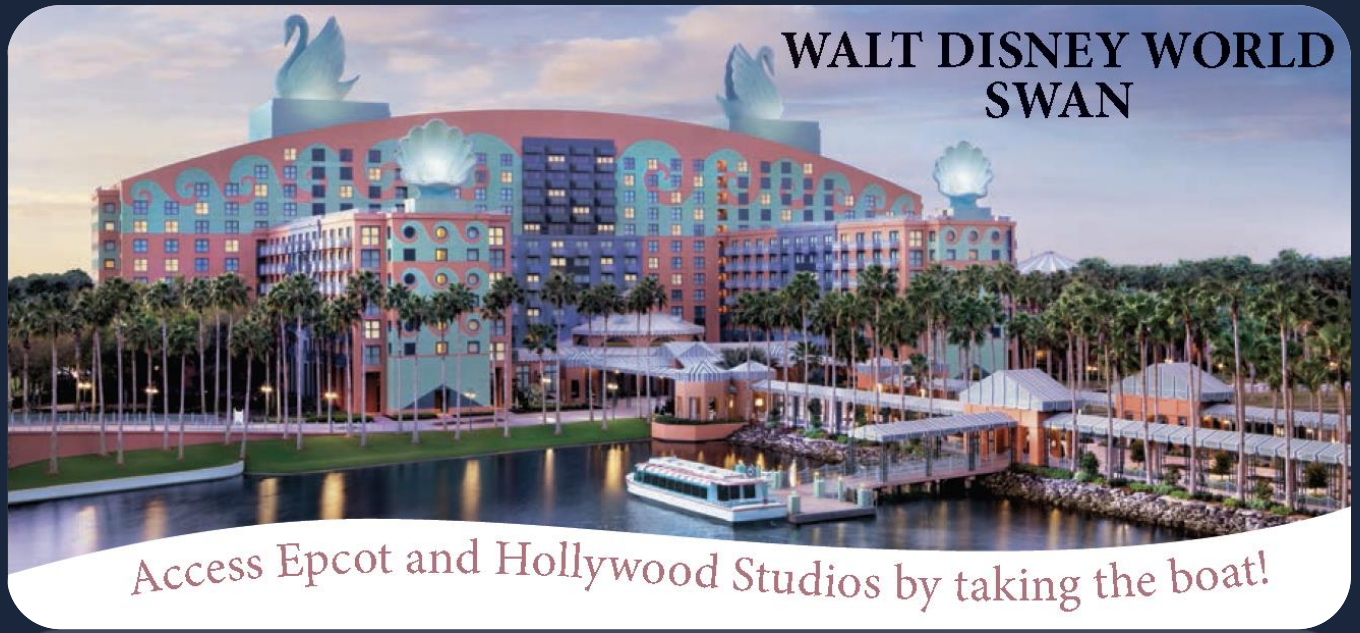
Your evaluation helps organizers build better conferences and helps speakers improve their sessions.

SQL *intersection*

Thank you!

Save the Date!

www.SQLintersection.com



2020

Week of April 6

We're back in Orlando!

SQL intersection

Leave the every day behind and enter a world of wonder and enchantment at the Walt Disney World® Resort. Located in the heart of the most magical place on earth, the Walt Disney World Swan and Dolphin Resort provides a truly extraordinary backdrop for our event! Beautiful tropical landscaping, tranquil waterways, and classic art and architecture work together to create a stunning landmark!