International Text in SQL Server

International Considerations

- Aim: supporting the storage and manipulation of multilingual data
- 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
- Issues:
  - Non-ASCII character representation
  - Date and time
  - Sorting and comparison of strings

Locale

- A locale is a set of information associated with a place or a culture—the name and identifier of the spoken language, the script used to write the language, and cultural conventions.
- SQL Server supports all 135 locales supported by Windows.
- Among them are five Chinese language locales (Hong Kong SAR, Macau SAR, the People's Republic of China, Singapore, and Taiwan); thirteen English language locales (Australia, Belize, Canada, Caribbean, Ireland, Jamaica, New Zealand, Philippines, South Africa, Trinidad, the United Kingdom, the United States of America, and Zimbabwe); and six French language locales (Belgium, Canada, France, Luxembourg, Monaco, and Switzerland).

Four different locales

Locale	English (U.S.A.)	French (France)	Japanese	United Arab Emirates
Country/region	United States	France	Japan	United Arab Emirates
Language	English	French	Japanese	Arabic
Written scripts	Latin	Latin	Kana, kanji	Arabic
Reading order	Left to right	Left to right	Left to right	Right to left
Windows-defined code page	1252	1252	932	1256
Time format	1:00 pm	13:00	13:00	1:00 p
Calendar	Gregorian	Gregorian	Gregorian (Localized)	Gregorian (Localized)
Default paper size	U.S. Letter	A4	A4	A4
Decimal separator	.	,	.	,
List separator	,	;	,	;
Thousands separator	,	space	,	,

International Transact-SQL

- Guidelines for making Transact-SQL statements more portable from one language to another:
  - When you perform month and day-of-week comparisons and operations, use the numeric date parts instead of the name strings. Different language settings return different names for the months and weekdays. For example, DATENAME(MONTH,GETDATE()) returns May when the language is set to U.S. English and returns Mai when the language is set to German. Instead, use a function such as DATEPART that uses the number of the month instead of the name. Do not code any logic that depends on the displayed names being from a specific language.

International Transact-SQL

- When you specify dates in comparisons or for input to INSERT or UPDATE statements, use constants that are interpreted the same way for all language settings:
  - ADO, OLE DB, and ODBC applications should use the ODBC timestamp, date, and time escape clauses of:
    { ts 'yyyy**-***mm***-***dd hh*:*mm*:*ss*[**.***fff*] **'}** such as:
    { ts '1998**-**09**-**24 10**:**02**:**20**' }**
    { d '*yyyy***-***mm***-***dd***'}** such as: **{ d '**1998**-**09**-**24**' }**
    { t '*hh*:*mm*:*ss***'}** such as: **{ t '**10:02:20**'}**

International Transact-SQL

- Applications that use other APIs, or Transact-SQL scripts, stored procedures, and triggers should use the CONVERT statement with an explicit style parameter for all conversions between the **datetime** and **smalldate** data types and character string data types. For example, the following statement is interpreted in the same way for all language or date format connection settings:

```
SELECT *
FROM AdventureWorks.Sales.SalesOrderHeader
WHERE OrderDate = CONVERT(DATETIME, '19960719',
   101)
```

Non-ASCI character representation

- In order to represent characters outside the 128 that are defined by ASCII there are two options
  - Using code pages
  - Using Unicode

Code Pages

- A code page is an assigment of international characters to the bit configurations of one or two bytes
- Most code pages require one byte, a few require two
- Code pages define the meaning of the bytes in **char**, **varchar** and **text**
- Depending on the chosen code page, the strings in char, varchar and text are interpreted differently
- **Code page** is the traditional IBM term used for a specific character encoding table
- The term code page originated from IBMs EBCDIC mainframe systems
- The IBM PC code pages are also known as the OEM code pages or the windows ansi code pages

Code Pages

- Each European language, such as German or Spanish, has its own single-byte code page. The bit patterns used to represent the Latin alphabet characters A through Z are the same for all the code pages, but the bit patterns used to represent accented characters vary from one code page to the next.
- Single-byte character sets cannot store all the characters used by many languages. Some Asian languages have thousands of characters; therefore, they must use 2 bytes per character. Code pages have also been defined around them. They actually use a mixture of single-byte and double-byte widths

Code pages that SQL Server 2005 supports

Code page	Description	Number of bytes
1258	Vietnamese	1
1257	Baltic	1
1256	Arabic	1
1255	Hebrew	1
1254	Turkish	1
1253	Greek	1
1252	Latin1 (ANSI)	1
1251	Cyrillic	1
1250	Central European	1
950	Chinese (Traditional)	2
949	Korean	2
936	Chinese (Simplified)	2
932	Japanese	2
874	Thai	1
850	Multilingual (MS-DOS Latin1)	1
437	MS-DOS U.S. English	1

Unicode

- The strings in nchar, nvarchar and ntext are interpreted as Unicode characters
- The Unicode UCS-2 encoding scheme is used and cannot be changed.
- Under this mechanism, all Unicode characters are stored by using 2 bytes.
- Unicode string constants are specified in SQL scripts with a leading N: N'A Unicode string'.

Unicode

- Storing data in multiple languages within one database is difficult to manage when you use only character data and code pages.
- For example, if we have to store
  - 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
- We must find a code page that will handle the characters of all three languages

Unicode

- You must also take care to guarantee the correct translation of characters from one of the languages when read by clients running a code page for another language.
- The easiest way to manage character data in international databases is to always use the Unicode **nchar**, **nvarchar**, and **ntext** data types, instead of their non-Unicode equivalents, **char**, **varchar**, and **text**.
- Because Unicode is designed to cover all the characters of all the languages of the world, there is no need for different code pages to handle different sets of characters.

Unicode

- If all the applications that work with international databases also use Unicode variables instead of non-Unicode variables, character translations do not have to be performed anywhere in the system. Clients will see the same characters in the data as all other clients.
- SQL Server stores all textual system catalog data in columns having Unicode data types. The names of database objects, such as tables, views, and stored procedures, are stored in Unicode columns. This enables applications to be developed by using only Unicode, and helps avoid all issues with code page conversions.

Unicode Standard

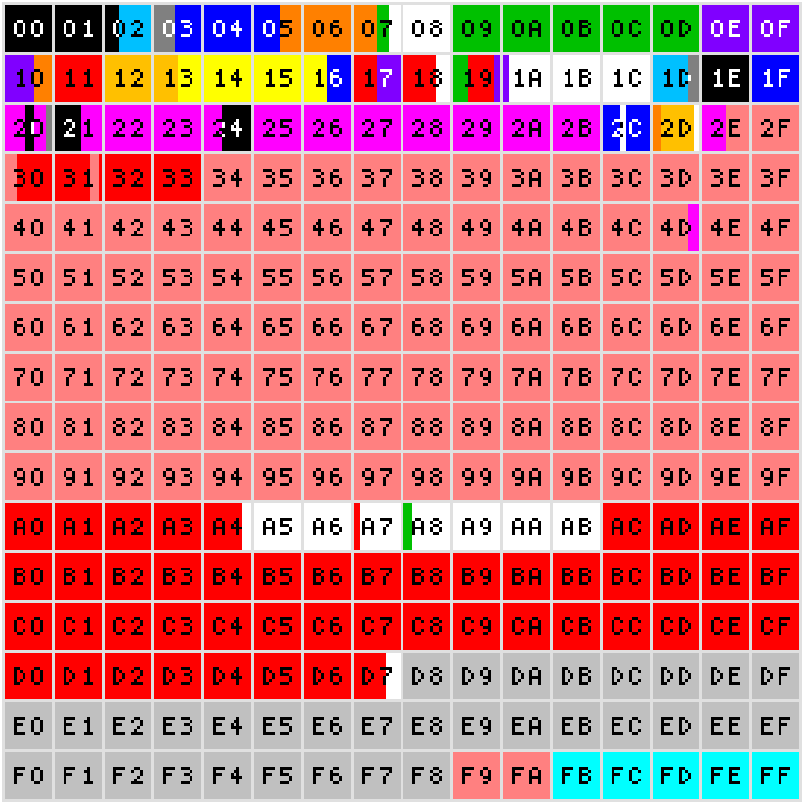
- **Unicode** is an industry standard designed to allow text and symbols from all languages to be consistently represented and manipulated by computers.
- SQL Server supports the Unicode Standard, Version 3.2, published in 2002.
- The UCS-2 encoding contains most characters widely used in businesses around the world.
- Differently from other Unicode encodings, the character codes have a fixed length of two bytes

Unicode Standard

- Unicode reserves 1,114,112 (= $2^{20} + 2^{16}$) **code points**, and currently assigns characters to more than 96,000 of those code points. The first 256 codes correspond with those of ISO 8859-1, the most popular 8-bit character encoding in the Western world. As a result, the first 128 characters are also identical to ASCII.
- The Unicode code space for characters is divided into 17 *planes*, each with 65,536 (= 2^{16}) code points.
- The first plane (plane 0), the *Basic Multilingual Plane* (BMP), is where most characters have been assigned so far. The BMP contains characters for almost all modern languages, and a large number of special characters. Most of the allocated code points in the BMP are used to encode Chinese, Japanese, and Korean (CJK) characters.

Basic Multilingual Plane

Black = Latin scripts and symbols
Light Blue = Linguistic scripts
Blue = Other European scripts
Orange = Middle Eastern and SW Asian scripts
Light Orange = African scripts
Green = South Asian scripts
Purple = Southeast Asian scripts
Red = East Asian scripts
Light Red = Unified CJK Han
Yellow = Aboriginal scripts
Magenta = Symbols
Dark Grey = Diacritics
Light Grey = UTF-16 surrogates and private use
Cyan = Miscellaneous characters
White = Unused



Each numbered box represents 256 codepoints.

Unicode Encodings

- Unicode defines two mapping methods:
  - the **UTF** (**Unicode Transformation Format**) encodings
  - the **UCS** (**Universal Character Set**) encodings
- The encodings include:
  - UTF-7 — a relatively unpopular 7-bit encoding, often considered obsolete
  - UTF-8 — an 8-bit, variable-width encoding
  - UCS-2 — a 16-bit, fixed-width encoding that only supports the BMP
  - UTF-16 — a 16-bit, variable-width encoding
  - UCS-4 and UTF-32 — functionally identical 32-bit fixed-width encodings
  - UTF-EBCDIC — an unpopular encoding intended for EBCDIC based mainframe systems

Storage Effects of Unicode

- All non-East Asian languages and the Thai language store non-Unicode characters in single bytes. Therefore, storing these languages as Unicode uses two times the space that is used specifying a non-Unicode code page.
- On the other hand, the non-Unicode code pages of many other Asian languages specify character storage in double-byte character sets (DBCS). Therefore, for these languages, there is almost no difference in storage between non-Unicode and Unicode.
- The non-Unicode code pages that specify character data storage in double-byte character sets are: Simplified Chinese, Traditional Chinese, Japanese and Korean.

Unicode Best Practices

- Storing character data in a specific code page may make sense if both of the following are true:
  - Conserving storage space is an issue, because of hardware limitations. Or, you are performing frequent sorts of lots of data, and testing indicates that a Unicode storage mechanism severely affects performance.
  - You are sure the code pages of all clients accessing this data match yours, and that this situation will not unexpectedly change.

Collations

- Collations specify the rules for how strings of character data are sorted and compared, based on the norms of particular languages and locales.
- For example, in an ORDER BY clause, an English speaker would expect the character string 'Chiapas' to come before 'Colima' in ascending order. But a Spanish speaker in Mexico might expect words beginning with 'Ch' to appear at the end of a list of words starting with 'C'.
- The Latin1_General collation will sort 'Chiapas' before 'Colima' in an ORDER BY ASC clause, while the Traditional_Spanish collation will sort 'Chiapas' after 'Colima'.

Collations

- When a collation is specified for non-Unicode character data, such as **char**, **varchar**, and **text** data, a particular code page is associated with the collation. For example, if a **char** column in a table is defined with the Latin1_General collation, the data in that column is interpreted and displayed by SQL Server using the code points of the 1252 code page.
- Collations specified for Unicode data, such as **nchar**, **nvarchar**, and **ntext**, do not have specific code pages associated with them, because Unicode data handles virtually all characters of all the world's languages.

Collations

- SQL Server provides two groups of collations: Windows collations and SQL collations.
- Windows collations are collations defined for SQL Server to support Windows locales.
- SQL collations are for backward compatibility only
- By specifying a Windows collation for SQL Server, the instance of SQL Server uses the same code pages and sorting and comparison rules as an application that is running on a computer for which you have specified the associated Windows locale. For example, the French Windows collation for SQL Server matches the collation attributes of the French locale for Windows.

Windows Collations

- There are more Windows locales than there are SQL Server Windows collations.
- The names of Windows locales are based on a language and territory, for example, French (Canada). However, several languages share common alphabets and rules for sorting and comparing characters. For example, 33 Windows locales, including all the Portuguese and English Windows locales, use the Latin1 code page (1252) and follow a common set of rules for sorting and comparing characters. The SQL Server Windows collation, based on the Latin1_General code page and sorting rules, supports all 33 of these Windows locales

Collations

- SQL Server collations can be specified at any level:
  - Instance, Database, Column, Expression
- When you install an instance of SQL Server, you specify the default collation for that instance.
- Each time you create a database, you can specify the default collation used for the database. If you do not specify a collation, the default collation for the database is the default collation for the instance.
- Whenever you define a character column, variable, or parameter, you can specify the collation of the object. If you do not specify a collation, the object is created by using the default collation of the database.

Windows Collations Names

<Windows_collation_name>::=
*CollationDesignator*_<ComparisonStyle>

<ComparisonStyle>::= *CaseSensitivity_AccentSensitivity [_KanatypeSensitive* [_*WidthSensitive*]] | { **_BIN | _BIN2** }

- *CollationDesignator:* Specifies the base collation rules used by the Windows collation. The base collation rules cover the following:
  - The alphabet or language whose sorting rules are applied when dictionary sorting is specified
  - The code page used to store non-Unicode character data.
  - Some examples are:
    - Latin1_General, French, Turkish

Windows Collations Names

- *CaseSensitivity*
  - **CI** specifies case-insensitive, **CS** specifies case-sensitive.
- *AccentSensitivity*
  - **AI** specifies accent-insensitive, **AS** specifies accent-sensitive.
- *KanatypeSensitive*
  - **Omitted** specifies kanatype-insensitive, **KS** specifies kanatype-sensitive.
- *WidthSensitivity*
  - **Omitted** specifies width-insensitive, **WS** specifies width-sensitive.
- **BIN, BIN2**: Specify a binary sort order

Windows Collations Names Examples

- **Latin1_General_CI_AS :** Collation uses the Latin1 General dictionary sorting rules, code page 1252. Is case-insensitive and accent-sensitive.
- **Estonian_CS_AS** : Collation uses the Estonian dictionary sorting rules, code page 1257. Is case-sensitive and accent-sensitive.
- **Latin1_General_BIN** : Collation uses code page 1252 and binary sorting rules. The Latin1 General dictionary sorting rules are ignored.

Sorting

- Kana-sensitive (_KS): Distinguishes between the two types of Japanese kana characters: Hiragana and Katakana. If this option is not selected, SQL Server considers Hiragana and Katakana characters to be equal for sorting purposes.
- Width-sensitive (_WS): Distinguishes between a single-byte character and the same character when represented as a double-byte character. If this option is not selected, SQL Server considers the single-byte and double-byte representation of the same character to be identical for sorting purposes.

Sorting

- For Windows collations, the **nchar**, **nvarchar**, and **ntext** Unicode data types have the same sorting behavior as **char**, **varchar**, and **text** non-Unicode data types.
- For binary collations on Unicode data types, the locale is not considered in data sorts. For example, Latin_1_General_BIN and Japanese_BIN yield identical sorting results when used on Unicode data.

Some Windows Collations

Windows System Locale	LCID (Locale ID)	Default Collation	Code page
Chinese (PRC)	0x804	Chinese_PRC_CI_AS	936
Chinese (PRC)	0x20804	Chinese_PRC_Stroke_CI_AS	936
Dutch (Belgium)	0x813	Latin1_General_CI_AS	1252
Dutch (Netherlands)	0x413	Latin1_General_CI_AS	1252
English (India)	0x4009	Latin1_General_CI_AS	1252
English (United Kingdom)	0x809	Latin1_General_CI_AS	1252
English (United States)	0x409	SQL_Latin1_General_CP1_CI_AS	1252
French (Belgium)	0x80c	French_CI_AS	1252
French (France)	0x40c	French_CI_AS	1252
German (Germany)	0x407	Latin1_General_CI_AS	1252
Greek	0x408	Greek_CI_AS	1253
Gujarati (India)	0x447	Indic_Genelal_90	Unicode
Italian (Italy)	0x410	Latin1_General_CI_AS	1252
Japanese	0x411	Japanese_CI_AS	932
Japanese (Unicode)	0x10411	Japanese_Unicode	932
Korean (Extended Wansung)	0x0412	Korean_Wansung_CI_AS	949
Spanish (Mexico)	0x80a	Modern_Spanish_CI_AS	1252