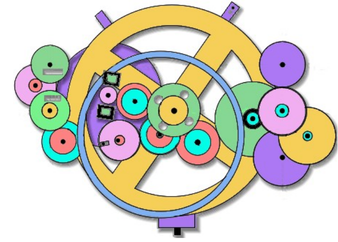
Antikythera Publications



DATABASE DESIGN NOTE SERIES

Relational Database Design
http://www.AntikytheraPubs.com
sweiss @ AntikytheraPubs.com

Exploring Alphabets
Multi-script Database Series #1

Prepared by: S. L. Weiss and F. Oberle

This is the first in a series of Database Design Notes intended to provide database designers with the background needed to address the issues that ***will arise*** when designing or converting an existing schema to support multi-lingual/multi-script data.

This Design Note could have been titled ‘Languages, Dialects & Countries, Writing Systems, Scripts, Abjads, Abugidas & Alphabets, Key Codes & Scan Codes, Character Codes and Character Cells, Typefaces, Glyphs & Fonts’ but that would have been unwieldy. But an awareness that such categories exist, and an ability to distinguish among them, is necessary for anyone implementing such a globalized system.

In the customer tables of such a multi-lingual/multi-script database, we might see given names such as Thomas, آمنة , ณัฐพงษ์, or שלמה, and possibly even recognize these as made up of Latin, Arabic, Thai, and Hebrew characters respectively. While it isn’t necessary to know that the latter three names – in Latin script – would be Aminah (a Saudi female), Nattapong (a Thai male), and Schlomo (an Israeli male), it is very important to understand the different characteristics that some writing symbols may have and, most importantly, the ramifications of their use to your database schema, supporting applications, report generation and other input/ output utilities.

Revised for public distribution: 22 December 2016 (with minor revisions in 2019)

See page 18 for information on other material from Antikythera Publications.

Copyright © 2016 by the Authors and Antikythera Publications

Permission is granted to distribute unaltered copies of this document, so long as this is not done for commercial purposes.

www.AntikytheraPubs.com

Database Design Notes about Multi-Language/Multi-Script Database Considerations

1. **Exploring Alphabets**
2. Exploring Complex Text Layout
3. Exploring UTF-8
4. Evaluating Fonts for use in Multi-Lingual Documents
5. Exploring Bidirectional Text Entry
6. Exploring Arabic Script Behavior
7. Exploring Han Script Behavior in Chinese

Let's start at the very beginning
A very good place to start
When you read you begin with A, B, C ...

Oscar Hammerstein II, from “The Sound of Music”

Database Design Note Series – Exploring Alphabets

PREFACE

It must be stated that although there is ample 'expert' support for each of the descriptions and opinions expressed in this paper, contrary 'expert' opinions can just as easily be found. These notes are intended only as a starting point for your own explorations of any segment you feel may be relevant. The key is to realize that, although many of the terms we regularly use seem to be treated almost as synonyms in "the West," the differences are not at all subtle when dealing with "non-western" languages and writing systems. While simply being aware that such differences exist provides a significant advantage when undertaking the design of (or conversion to) an internationalized data schema, knowing just a bit more about each of the terms described here will greatly assist in navigating the many changes involved.

Languages, Dialects & Countries

A Language is a method of human communication consisting of meaningful sounds or symbols that are more or less structured by a generally accepted if not always formalized grammar. A Language may be used in aural (e.g. spoken), visual (e.g. printed or displayed), or tactile (e.g. Braille) fashions, alone or in a variety of combinations. Some ancient languages – ones that specialists have discovered how to read but not pronounce – have been unspoken for perhaps thousands of years, but are still written. Any language is presented using at least one Writing System, and several languages can be and are written with several different "Scripts" – one of the terms defined a little later.

The International Standards Organization has produced a set of documents – imaginatively named ISO 630-1 through ISO 630-5[1] that establish two and three character Latin alphabetic codes for identifying the world's Languages. This seems like a great idea, except that a) these standards deal only with Living Languages, and b) there has never been any consensus on the differences between Dialects and Languages (e.g. when does a dialect qualify as a separate language?). The only generally agreed-upon taxonomy seems to show 'Language Families' as the universe of discourse – with each Language Family containing multiple child Languages – and each child Language possibly containing a variety of Dialects. These categorizations, as might be expected for living languages, cannot be considered static.

Fortunately, for the language-related functionality specific to office applications, such as spell checking, thesaurus use, and grammar checking,[2] the definition of Language – whether living or dead, real or artificial – can often be inferred if need be from the existence of a dictionary or similar support file(s).

Country is often a rather transient concept; many 'countries' have existed for thousands of years, but their borders, languages, cultures, scripts and even ethnic compositions may have changed dramatically over time due to political shuffles. Country names, though, are commonly used as a convenience to distinguish among what are really broad dialects of common languages, such as the sixteen or more 'English (*xxx*)' Languages presented as options in much software.[3]

Writing Systems

Although this term is commonly used and understood, the word 'writing' suggests that it refers literally to something 'written' or 'displayed,' but in reality the term also encompasses all the visual and tactile Language representations discussed earlier.

It is generally supposed that Writing Systems began as collections of simplified drawings of objects and, later, of concepts and/or sounds. The first symbol in Figure 1, for instance, represented 'an ox' while a logograph[4] depicting two or three oxen represented the concept of a 'herd' rather than a specific quantity.

---

1 Until 2014, there was an ISO 639-6, but that was dropped; apparently, even for bureaucrats, five was considered plenty.

2 Soundex Codes, specific to English names (although parallels exist for some other languages), also fall into this category.

3 Standardized country codes include ISO 3166-1, 3166-2, and 3166-3. Another source for related data can be located at the United States CIA's site https://www.cia.gov/library/publications/the-world-factbook/appendix/appendix-d.html.

4 A logograph (also called a logogram) is a single written character or glyph that represents a word or even an entire phrase.

These symbols later began to be used and understood in context simply as phonemes (syllable sounds) that formed longer spoken words. Symbols such as 'Sun' and 'Flower' might be merged to suggest the concept of a sunflower. Today we might call this 'scope creep' but such increasingly arcane and obscure symbol combinations enhanced the job security of the few scribes who were able to keep up with them, so this type of writing system continued to be used.

Eventually, though, the benefits of using symbols to represent more discrete sounds began to drive the development of what we know today as Alphabets;[5] this transitional process is illustrated (but a bit over-simplified) in Figure 1 below.

The early Phoenician icon for an Alep (Ox) is shown on the far left of Figure 1. Eventually, that symbol came to represent the Alep *sound* as well, rotating a little over -90° in the process, as shown in the middle symbol. After some further rotation it became what we would now consider a 'letter' – the rightmost 'A' symbol – that survives in a variety of present-day alphabets. Although the physical similarities aren't immediately evident, the name 'Alep' survives in character names such as Hebrew's *Aleph* (א) and Greek's *Alpha* (the capital A and its small – sometimes known as lower case – version α).

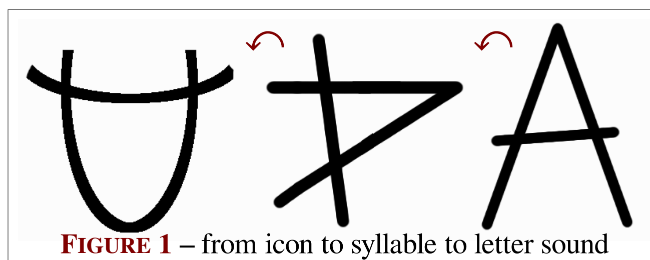


FIGURE 1 – from icon to syllable to letter sound

All these 'systems' of writing remain in use today, not only in academic studies of ancient texts, but by any potential users of report writers, text editors, word processors and – more to the point – databases. Chinese logographic characters (known as hànzì) dating from at least the 5th century are still in use, and were adopted by the Japanese – by whom they are known as Kanji. Japanese is also written using two sets of phonetic symbols known as Hiragana (sounds used in native Japanese words) and Katakana (sounds used in words borrowed from other languages).

Additional special-use writing systems have been designed for the needs of particular groups; examples include Braille, the International Phonetic Alphabet system and various clerical, legal, and medical shorthand notations.

Each writing system has a number of characteristics, although only one – directionality – is relevant to this paper. Symbols in a given writing system can be laid out in the familiar left-to-right/top-to-bottom and right-to-left/top-to-bottom directions[6]; Other layout patterns include top-to-bottom/left-to-right, top-to-bottom/right-to-left, bottom-to-top in either direction,[7] several Boustrophedon hybrids (alternating left-to-right/right-to-left patterns that get their name from the path taken by the aforementioned Alep (ox) as a field was plowed),[8] and even spiral patterns such as that shown in Figure 2.

More detailed discussions of the world's variety of Writing Systems can be located easily, but the important things to note here are:

- A Language may use one or more Writing Systems.
- A Writing System may be used by many Languages.
- Writing Systems are not necessarily alphabetic.

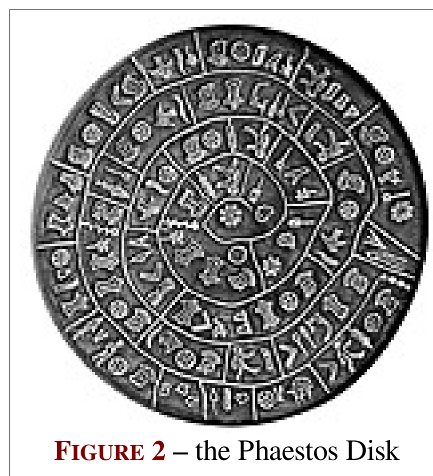


FIGURE 2 – the Phaestos Disk

And remember, we haven't even considered sequences of drum beats, Morse Code, hole patterns on cards and paper tape, little-endian and big-endian bit sequences in a variety of lengths both fixed and variable, and the tactile Braille. Assuming the availability of appropriate printing and presentation/editing devices, there is no reason why such tactile writing systems should not be supported by a word processor. Support for Boustrophedon or spiral layouts, however, will likely remain the responsibility of graphics software for some time to come.

5 With fewer symbols required to represent speech, the relative ease of learning Alphabets encouraged a rise in literacy.
6 Known rather bizarrely by much software – including Microsoft Office and LibreOffice – as 'normal' and 'RTL' respectively.
7 The two top-to-bottom layouts are referred to inadequately as 'Asian,' while the others don't seem to be supported at all.
8 Illustrations of Boustrophedon and Reverse Boustrophedon are given in Exploring Complex Text Layout, another Design Note in this series that is also available from www.AnitkytheraPubs.com/i18n.htm.

Scripts

Some sources make no distinction between Writing Systems and Scripts. For what it's worth, that seems to me to be an untenable position, particularly when organizing data management and software functionality.

In order to standardize the identification and transmission of symbols used to represent the world's writing systems, the Unicode Consortium took on the task of identifying each and every symbol ever used for that purpose, and giving each of them a unique number. Even more helpfully, they began organizing these symbols into 252 related groups called Blocks such as the alphabetic symbols we'll be using in this series: Hebrew U+0590-05FF, Devanagari U+0900-097F, and Thai U+0E00-0E7F, for example. Examples of symbols from other writing systems include the IPA[9] 0250-02AF, Braille U+2800-28FF, Egyptian Hieroglyphics U+13000-1342F, and even Shorthand U+1BC00-1BC9F and some versions of American Sign Language e.g. U+1D800-1DAAF. A particular Script uses characters from one primary Block, but may be (and often is) supplemented by characters (e.g. punctuation, variant character forms, etc.) from other blocks.

Some Unicode blocks contain symbols that are intended as supplements for specific scripts, while others contain symbols that are not part of any particular alphabet. Such blocks may contain mathematical symbols, various types of diacritic and other marks, composite characters such as ligatures, and so forth.

These Unicode blocks are further grouped into larger sections such as European Scripts, East Asian Scripts etc. but this is *solely for convenience* since many writing systems and languages utilize symbols from more than one of these symbol groupings.[10] Although any of the character Scripts/Blocks may sometimes be equivalent to all or part of a particular language's alphabet, they should not be viewed as alphabets themselves. The distinction will hopefully be clarified in the next section. For purposes of this paper, the distinction between Block and Script is not relevant.

Some Scripts have obvious commonalities with others, but this cannot be taken to mean there is any relationship at all between the languages that use those scripts. Cyrillic script, for instance, was originally derived from the Greek for recording the Russian language, and supplemented with some additional symbols (e.g. Ж, Ц, Я, among others) for sounds the Greeks didn't use. Linguistically, however, the Greek and Russian languages have nothing in common.

The Cyrillic Script was itself eventually adopted as the basis of the alphabets used in a variety of neighboring languages.[11] Surprisingly to many, it is also used in alphabets developed for some Eskimo-Aleut dialects and languages spoken in present-day United States. Generally, alphabets are associated as much with Languages as with Scripts, suggesting some relevance to word processing applications. But things can seem stranger.

Many assume that because the Thai language uses Thai Script, the Thai Language is *always* written with Thai Script. Printing Thai with Cyrillic Script might therefore seem rather unlikely, but consider a Russian traveler preparing for a visit to Bangkok who might purchase a book of useful Thai phrases. Because most non-Thais (including database designers and software developers) are likely not familiar with Thai Script, nor have time to learn it, phrases in a Russian guidebook are often rendered – however imprecisely – in Cyrillic Script. Even more interesting, in guidebooks for an Egyptian traveler, the Thai phrase would be presented with Arabic characters running right-to-left, even though Thai is written, like English, from left-to-right.

Furthermore, many languages and scripts share common separators (such as the space and tab), punctuation marks, hyphens, paired delimiters (e.g. parentheses, brackets and the like) and numeric characters from the Latin block formerly known as 'lower ASCII.'[12] There are also blocks containing various types of unique punctuation, as well as a variety of graphical symbols.[13] The bottom line: there is no one-to-one correspondence between Scripts and Languages.

---

9 The International Phonetic Alphabet, which isn't associated with any particular Language.

10 See http://www.unicode.org/charts/ for listings. Particularities for each Language using the Block/Script are shown as well.

11 In addition to Russian, these include Belarusian, Bosnian, Bulgarian, Karelian, Kildin Sámi, Komi-Permyak, Kurdish, Macedonian, two Mari, one Mongolian, Montenegrin, Ossetian, some Romani, Rusyn, one Serbian, Tajik, and Ukrainian alphabets. Although similar, a variety of characteristics, e.g. alphabet composition and sorting order can be seen across this group.

12 The Design Note "Exploring Bidirectional Text Entry" provides examples of not accounting for this symbol sharing across Script blocks. Handling these isn't always as straightforward as it might seem, but it does have a certain underlying logic.

13 Standard Script Codes are defined by ISO 15294 (see http://unicode.org/iso15924/iso15924-codes.html); more details about Unicode Scripts are available at https://en.m.wikipedia.org/wiki/Script_(Unicode).

Abjads, Abugidas & Alphabets

Most western Writing Systems such as Latin are based on the use of what are generically called 'alphabets,' but there are potentially relevant, and in some situations quite important, distinctions within even that category.

The earliest form of what we now call the 'alphabet' is generally known by its Arabic name Abjad (أبجد) and is still in use today. An Abjad indicated only the sounds we now call consonants. If an abjad were used to write English, we might see something like 'BJD' ('abjad') or 'LFBT' ('alphabet') where the reader would need to fill in some appropriate vowel sound simply to be able to pronounce the words at all.

Later, realizing that identical words with differing vowel sounds, such as 'TRBL' in the sentence 'TH CLRNT PLR HD **TRBL** GTNG CLR **TRBL** FRM TH RSTY NSTRMNT.'[14] might be subject to misinterpretation by a reader, certain markings began to be added over, under, or inside consonants to associate them with specific vowel sounds. These slightly more sophisticated form of "alphabets" – still not giving equal status to vowels – are now known rather harshly as 'impure abjads' and almost exclusively use vowel symbols that are much smaller than those of the consonants. Most abjads in use today, including Arabic, Hebrew, and Aramaic, are thus 'impure.' Even the word أبجد as written here is itself impure! Such ironies may explain why people mostly just use the term "alphabet" to refer to the blanket class.

Several modern Abjads add full-sized vowel characters to supplement their diacritic vowels. The presumed further decrease in purity of these Abjads doesn't appear to be discussed in the relevant literature. In Thai, for instance, there are both full-sized vowels such as โ, as well as vowels in both superscript (like ื) and subscript varieties (like ู) as well.

An Abugida[15], another early member of the "alphabet" class, began with languages having similar sounding 'filler' vowels; as the need arose to indicate multiple vowel sounds, this class – rather than adding separate diacritics to the consonants – made slight modifications to the consonants themselves which typically included attaching hooks or other appendages. In some cases, consonants were rotated to indicate different vowel sounds.[16]

Democracy, along with full equal rights and status for vowels, seems to have first been introduced by the Greeks, thus making classical Greek the first true alphabet in the sense we commonly think of today.

Such an Alphabet is a set of character-class symbols from a single Script that are used by a particular Language. Each Alphabet will have a defined order, so that lists may be sorted, dictionaries easily queried, and so forth. An Alphabet may not include all symbols from its Script, and one alphabet's prescribed ordering for one Language may not be the same as that of another Language using the same Script.

Although the Latin 'A,' the Greek 'Alpha,' and the Cyrillic 'Ah' – first letters in their respective alphabets – might appear to be 'shareable,'[17] and may all use what look like (and what may even be) identical glyph representations (A, Α, & А respectively in FreeSerif[18]), they are separate and distinct *characters*! Other characters that appear identical, such as the Latin 'N' (`U+004E`) and the Greek 'N' (`U+039D`) represent a consonant and vowel respectively. While certain classes of symbols may be shared across Languages, alphabetic characters are never shared across Scripts.[19]

---

14 Easy: 'The clarinet player had ***trouble*** getting clear ***treble*** from the rusty instrument.' Note that programmers may already be familiar with several Abjad examples without realizing it; 'CLR' for example is used in several command scripts and languages.

15 Abugida is pronounced with a hard "g" similar to that in the word "golf."

16 Really! Several writing systems and alphabets for Canadian aboriginal languages are constructed this way.

17 If you care, their distinct Unicode character points are `u+0041` (65d), `u+0391` (913d), `u+0410` (1040d) respectively.

18 But – as displayed here in the FreeSerif font, you many notice that the Greek A (0x0391) is slightly taller than the other two.

19 They are, however, used in URLs intended to draw people to impostor web sites: one of the few drawbacks to Unicode is the potential security risks this can present to an unsuspecting user. With the right font, the phishing site www.NewBank.com could easily be mistaken for the legitimate site www.NewBank.com, since the Greek (or Cyrillic) and Latin "N" characters are so similar. This is not intended as a security Design Note, but as your operation begins entering the multi-lingual world, IT personnel, particularly those involved with networking, should be aware of this potential. Depending on the type of data stored, it may also need to be considered when designing column constraints (e.g. URLs might only permit Latin characters).

Perspectives on Progress

FIGURE 3 – Early Word Processor, circa 3000 BCE
A heavier **Bold mallet** was an available option; *italic mallets* were not available though, since Italy hadn't yet been created to give that style a name!

Before continuing this section, an unfair look at the changes in technology over the past several thousand years is in order. Unlike the current incarnation of LibreOffice Writer, for example, early versions of its predecessor Libre-Gouger shown to the right had no restrictions on the variety of Scripts that could be mixed on a single stone.[20] To be fair, the editing capabilities (e.g. copy, cut, and paste) of those early versions remained quite minimal. (Still awake?)

If this discussion leaves the impression Writing Systems progressed steadily from pictures and icons through syllable sounds, characters, and finally to alphabets, it must be said that recent Unicode symbol additions include a bikini (`U+1F459` 👙), high-heeled shoe (`U+1F460` 👠), and computer mouse (`1F5AF` 🖯)! As near as we can determine, none of these were ever used by early Egyptians or Greeks.[21]

Key Codes & Scan Codes

English-speaking users likely refer to the keys of a computer keyboard with names like 'the Tab key' or 'the A key.' Thai-speaking users might actually use those same terms, but are just as likely to call the latter 'ปุ่ม ฟ' (meaning 'the ฟ key' or, more literally, 'the button ฟ').

For the most part, despite the accouterments of some specialized products, keyboards have no concept of Letters or Symbols; the key names simply reflect whatever is printed on the key tops. Keyboards are, at their core, a collection of anywhere from 40 to well over 100 switches – usually arranged in rows and staggered columns, and internally identified by their particular row-column intersections. Groupings of keys – often indicated by different coloring – are devoted to particular uses, such as alphanumeric, punctuation, cursor movement, editing, and so forth.

On a venerable Northgate OmniKey Ultra keyboard, the key marked with the 'A' operates switch #38.[22] When that switch is pressed, the keyboard sends the eight bit sequence 00011100 (decimal number 28 or the hexadecimal value `0x1c`) to the computer. When the key is released, the sequence 10011100 (156; `0x9c`) is sent. Such sequences originating within the keyboard are called Scan Codes.

Note that any key-down sequence differs from its corresponding key-up sequence in the first bit, meaning the key-up value is always equal to the key-down value plus 128; this assists the computer in keeping track of key presses that might not appear in sequence. When key #38 is released, and barring any other information being available, the computer uses its knowledge of the keyboard to interpret the 'key #38 down + up' sequence as the single byte 01100001 (97 `0x61`), the ASCII and Unicode value of the lower case 'a,' not the capital 'A.' Even with 'ฟ' or 'ฤ'[23] printed on the key, a Thai keyboard works the same, as do "standard" keyboards for other languages.

At this stage, the computer knows only about symbols represented by numbers between 0 and 127 – formerly called "lower ASCII" and now Unicode's Basic Latin range. At this low level, typical computers recognize only the Latin characters that English uses, since users and their languages are really not of much interest. Command lines, batch files and shell scripts, programming languages, and processor op codes were initially created by English speakers and there seem to be few if any benefits worth the effort to change this. Operating systems have matured to the point, however, that any language and script can be recognized at higher levels without users even being aware of this limitation.

---

20 The famous Rosetta Stone of 196 BCE, containing Hieroglyphic, Demotic, and Greek Scripts, is a good example of this.

21 One has to wonder: if the transition from logographs to alphabets represented the rise of literacy, does the current proliferation of Emojis (colored logographs?) indicate literacy's impending fall? More new Emoji are being added each year!

22 The internal scan codes used in your keyboard may be entirely different. See page 16 for means of identifying these.

23 Thai has 44 consonants along with a variety of vowels and tone markings, but has no concept of capital and small letters; these two characters are different and unrelated, with the less frequently used ฤ character entered by typing ฟ with the Shift key.

To see a simple example of how modifier keys such as Shift, Ctrl, Alt (or combinations of those) are used to interpret key presses differently, consider the following sequence of key presses using the Shift key:

Shift⇓	Switch #50 closed:	Bit sequence sent from the keyboard is `00010010` (18; `0x12`). Modifier key pressed[24] and active; the computer waits to see what's next.
A⇓	Switch #38 closed:	Bit sequence sent from the keyboard is `00011100` (28; `0x1c`). Character key pressed; but the 97 must be 'modified' (reduced by 32).
A⇑	Switch #38 opened:	Bit sequence sent from the keyboard is `10011100` (156; `0x9c`). Character key released; interpret the 'a' as 'A' (65) due to modifier. Modified value is forwarded for further interpretation or processing.
Shift⇑	Switch #50 opened:	Bit sequence sent from the keyboard is `10010010` (92; `0x146`). Modifier key released; modification buffering is stopped.

Not all modified key sequences are sent on to applications for further processing; the Ctrl+Alt+Del sequence made infamous by DOS and MS-Windows™ is acted upon immediately by the system itself. "*Key Codes & Scan Codes*" on page 7 provides a more detailed look at how these modifiers function below the covers, and how their activity can be examined in a system set up to support multiple Scripts and Languages.

Character Codes and Character Cells

This Note began by providing an alternate, but rather unwieldy, title. In keeping with that theme, this section could also be titled "What you see is not necessarily what you typed – and what you typed is not necessarily what is stored – and what is stored is not necessarily what is displayed – and what appears on screen or paper is not necessarily displayed in the order in which you typed it." This is partially related to the differences among Alphabets, Abjads, and Abugidas discussed earlier, but is compounded by IT history and inertia. A database designer dealing with multi-lingual, multi-script data needs to be generally aware of these occurrences. The details can always be looked up.

A Character *Code* is, as described earlier, the specific number assigned to represent a particular symbol; although "character" is often used colloquially in English as a synonym for 'letter,' that is limiting and thus inaccurate.

A Character *Cell* is what appears on screen or paper as an "imaginary" placeholder for a character; the distinction can be seen in Figure 4. The Thai snippet นี้ คือ ("this is") appears to (and does) occupy four Character Cells, but actually consists of seven characters that are stored separately on disk, each of which has its own symbol and code point.

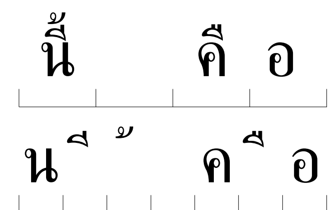


FIGURE 4 – Thai Script: 7 Characters displayed in 4 Cells

The initial consonant น has both vowel and tone mark diacritic characters, followed by a space, the consonant ค with a vowel diacritic, and a final, seventh, character อ. The importance of recognizing the distinction between characters and character cells cannot be overstated, and failure to do so can cause unexpected and potentially confusing results, particularly when sizing database columns for Unicode text storage; a `VARCHAR(6)`, for example, would be insufficient to hold this apparently four-character Thai phrase and cause an error.[25]

Linux's `m17n` library used a Thai Script font layout table (`THAI-GENERIC.flt`)[26] to reorganize the consonant-vowel-tone sequence into the composite glyphs *displayed* in the Character Cells shown in the top row of Figure 4. In this example, storage of the individual characters wasn't affected, as seen in the second row. The glyphs were then passed along to a rendering engine for display. Much is happening under the covers.

24 In this example, the left shift key is shown, but the results using the right shift key would be the same. What is important to note is, at a low level, each switch on a keyboard is independent of any other, and can be detected separately if desired.

25 Actually, there is more important reason that even a `VARCHAR(12)` can't hold it, but that also appears in "Exploring UTF-8".

26 See http://manpages.ubuntu.com/manpages/precise/man5/mdbFLT.5.html for a short description. On Ubuntu, these font layout table files are located in /usr/share/m17n/. Other systems are in use but script-specific font layout tables, which use something akin to regular expressions with glyph positioning as well as substitution variables, seem to give the best results, particularly since font-specific layout tables can be created where required to support decorative fonts as well as alternate Scripts.

When preparing for multi-lingual, multi-script implementations by the way, the organization's font portfolio must be considered, as the available fonts may affect the accuracy of any output, but that is out of scope for this paper.[27] On most systems, there are multiple methods to enter characters or symbols that aren't given their own dedicated keys. These are discussed in more detail in the next Design Note in this series.[28] For the moment we'll only deal with the Compose Key (Comp) method[29], and some word processor features you and your users are likely already aware of.

The table below will introduce some variants of Figure 4's "character-codes versus character-cells" theme:

As entered:	Comp o c 006F 0063	(c) 0028 0063 0029	Comp u " 0075 0022	Comp n ~ 006E 007E	1 / 2 0031 002F 0032	f f l 0066 0066 006C	ह ि 0939 093F
⇓ As stored:	© 00A9	© 00A9	ü 00FC	ñ 00F1	½ 00BD	f f l 0066 0066 006C	ह ि 0939 093F
⇓ As displayed:	© 00A9	© 00A9	ü 00FC	ñ 00F1	½ 00BD	ﬄ FB04	हि
	FIGURE 5.	**FIGURE 6.**	**FIGURE 7.**	**FIGURE 8.**	**FIGURE 9.**	**FIGURE 10.**	**FIGURE 12.**

FIGURE 5: This illustrates character substitution using the Compose key. Typing Comp, o, c on most systems will result in the symbol © not only being displayed, but replacing the o-c sequence in memory and on disk. Be aware, though, that Comp, c, o produces a different character (ǒ)! Many operating systems and applications also provide "Character Map" utilities that permit users to select characters such as © from a graphic interface. In word processors, such character map dialogs are often available by using a menu option such as "Insert > Special Character" or similar.

What makes this example different from Figure 4 above is that, in the Thai example, the changes are only on the display; the Thai characters all remain in memory and on disk. With character substitution, we can no longer edit or delete either parenthesis or the "c" character; they no longer exist! Understanding the difference is important.

FIGURE 6: Many word processors offer auto-correct features. In some of these, typing (c) will also enter the ©. Another auto-correct variant is to use delimited keywords that will cause the same substitution. In LibreOffice Writer, for instance, typing ":copyright:" (the word delimited with colons), achieves the same effect described in Figure 5.

Operating systems and applications often offer the ability for a user or (in some cases an administrator) to edit the key sequences defined for the Compose Key or Auto-Correct functions, but the methods for doing so vary too much for any guidelines to be offered here. Adding certain key combinations can have an impact on data integrity, however; the description of Ligatures (Figure 10 and its variants below) will illustrate an example of how this can occur.

Before proceeding to the example in Figure 7, we need to make mention of how strings are sorted.[30] Two questions will expose the issues involved: Should "March" always come before "march" in a sorted list? Does case even matter when sorting in a particular language? It would appear that the Oxford English Dictionary and Merriam-Webster's Collegiate Dictionary, both well-regarded *English* references, differ on this, with the latter placing "march" first. To get an idea of the complexity of this particular aspect of handling multi-language, multi-script data, it might be discouraging to read documentation such as Oracle's http://docs.oracle.com/cd/B19306_01/server.102/b14225/ch5lingsort.htm. Remember, though, that approaching such a project only requires an awareness that such issues exist; having that awareness along with access to the Internet and your own products' documentation[31], it is possible to succeed.

---

27 The fourth Design Note in this series, "Evaluating Fonts for use in Multi-Lingual Documents", will help assess these.
28 See the section "Character Entry Methods" in "Exploring Complex Text Layout" for a more detailed discussion.
29 The term "Multi-Key" is sometimes used instead of "Compose Key."
30 To sound suitably professional, we could have said we're discussing collation sequences, but this is an informal introduction!
31 Documentation seems to have fallen out of favor over the past few decades, since it is difficult and expensive to produce and, according to many vendors: "no one reads it anyway." For products that will be used in multi-lingual, multi-script systems, however, lack of good documentation or an acceptable substitute should be a cause for concern. This can get tricky!

FIGURE 7: (page 9) At first glance, this creation of the ü character – whether using a Compose key sequence as shown or using an auto-correct feature, would seem to be very similar to the creation of the © character in Figures 5 and 6, but there are additional considerations when dealing with combinations that result in alphabetic characters. On the title page of this document we referred to storing our customers' Names; although a single ü could be part of any other data element, we'll continue restricting the scope to names, and discuss how Anja Müller's Surname[32] might be handled.

The umlaut over the u (ü) suggests that our customer is German but, since umlauts are used in other languages, we can't be certain. And if she is German, we must still determine which particular flavor of German is her native Language. Assume for the moment, though, that we know she isn't Turkish (Türk) from other data we have (e.g. her address).

The German DIN 5007 standard defines two differing approaches to sorting based on what the list is used for. Entries in dictionaries and similar lists treat "u" and "ü" as if they were the same. In lists of names, however, "ü" would follow "ue." Thus Franz Mueller – not yet a customer – would always come before our customer Anja Müller. The same rule also applies to the "ä" character. This same use-case dependence on ordering is true for the Austrian version of German as well.

Swedish, however, which also uses the diaeresis, is sorted from A-Z as expected, with the characters Å, Ä, and Ö usually – but not always – following "Z." Since the lower case å, ä, and ö exist as well, so does the pesky choice of ordering by case.

Frivolous Aside: The two dots …

… above the ü are one specific form of *diaeresis* (also spelled *diæresis* or *dieresis*, and if you can't spot the difference between the first two spellings yet, you will be able to once you've read the example shown in Figure 10).

The *umlaut* (also known as a *tréma* in French – meaning perforation – the word used for the dots on sets of dice) changes the sound of the letter over which it is placed.

The identical symbol placed over words such as naïve and coöperate indicates that the character is not joined with the previous character but pronounced as a separate syllable. These are more properly known by the generic term *diaeresis* (however you spell it), but are not commonly seen in modern American English writing.

If you were wondering, the *single* dot over the Latin letter i is called a *tittle*. More about that interesting artifact later!

Because we enjoy rants, links to one of these as well as another explanatory note are provided in Footnote 33.

These issues cannot be resolved without knowing the specific design aims (i.e. the Countries and Languages targeted by your marketing department, who likely didn't share those aims with you ahead of time) but, again, being aware that these issues exist will help. Remember, of course, that after adding Frau Müller[34], we need to decide how to properly sort our now extended list of customers from 5 countries and 4 Scripts – two of which are right-to-left.

Fortunately, default collation sequences can be set not only for an entire system, but independently (and differently) for a single database instance in most RDBMS products, many of which further permit these to be specified explicitly on a per-column basis in SQL queries (perhaps based on customer attributes like home country or even province).

FIGURE 8: (page 9) The "ñ" character presents a slightly different situation. Some apparently 'composite' characters such as the Spanish ñ qualify as independent alphabetic characters even though we can still enter them on Latin keyboards using any of the techniques mentioned above. The ñ is listed as a full-fledged character in the Spanish alphabet (after the n) with none of the equivocations noted for German's umlauts and its ß character.[35] When Spanish words are sorted, the ñ follows the n in all cases. With a Spanish keyboard[36], the ñ has its own dedicated key that produces Ñ when shifted – no special Compose Key sequences are needed.

FIGURE 9: (page 9) Character replacement in examples of this type aren't cleanly divided into "always replaced" or "only combined for display purposes." Eighth fractions (⅛, ¼, ⅜, ½, ⅝, ¾ and ⅞) can often be entered using any of the methods described above, and result in character *replacement* as with the ©. On the other hand, techniques for entering fractions such as $^{1}/_{3}$ and $^{7}/_{16}$ are different (out of scope here), and these are *not* replaced by a single character.[37]

32 A look at https://www.thegermanz.com/how-germans-feel-umlauts-aou/ will explain why we chose Frau Müller!

33 The rant can be found at http://www.newyorker.com/culture/culture-desk/the-curse-of-the-diaeresis, while the more serious commentary is at http://www.grammarphobia.com/blog/2011/04/diaeresis.html.

34 … who, as it turns out, is actually from Melbourne, Australia. Sort that!

35 A typical description of the German alphabet says ä, ë, ö, ü and ß "… do not constitute distinct letters in the alphabet."

36 A real, physical keyboard or many of the keyboard layouts used with Input Method Editors. (Ñ is located where our **;** key is.)

37 Think about it. Even Unicode can't accommodate dedicated glyphs for every possible fraction – not even $99^{44}/_{100}$'s of them!

Another impediment to our never-ending quixotic quest for linguistic consistency is that the 'æ' character is viewed as an actual alphabetic character in Denmark, Iceland, and Norway, but as a ligature in others. Words with some ligatures are recognized by some word processor dictionaries[38]; some words are automatically 'expanded' before being passed to spell checking routines; some simply fail the spell check. In short, there is still a wild-west quality in this area.

FIGURE 10: (page 9) In printing, a Ligature is – loosely – a combination of letters/glyphs used in professionally typeset material. The example of a Ligature[39] given in Example 10, although apparently similar to the previous examples, can actually be a source of some difficult-to-trace issues that arise in many systems, not only with data, but with documents of all types – including reports based on *your* data. When the f-f-l sequence is typed in some of your applications, you can't always predict if the result will simply be shown as individual letters or as an ﬄ Ligature. A bit of informal background for non-typographers may help.

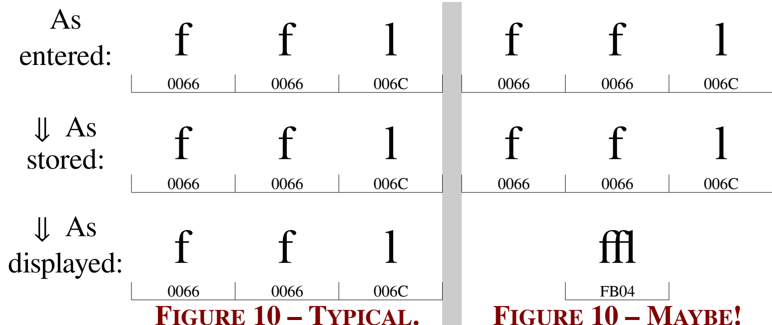


FIGURE 10 – TYPICAL. FIGURE 10 – MAYBE!

FIGURE 11: Professionally type-set documents are certainly not the province of a database designer, but those who wish to achieve publication quality output (perhaps the marketing department making use of "your" data) might wish to include Ligatures in their printed material. Examples commonly seen in English include æ, ﬀ, ﬁ, ﬂ, Figure 10's ﬄ, and even various artistic variants like ﬅ. Thus aesthetic might be printed as æsthetic, efficiency as eﬃciency, finished as ﬁnished, fluid as ﬂuid, and waffle as waﬄe. The differences in appearance when these Ligatures are used can be more easily seen in Figure 11. In the case of "waffle," it is easy to see how the ligature removes those annoying gaps between the two "f"s and the "f" and "l" no matter how well the font is kerned.

(11A) – FIGURE 11 – (11B) (W/O LIGATURES)	(WITH LIGATURES)
aesthetic	æsthetic
efficiency	eﬃciency
finished	ﬁnished
fluid	ﬂuid
waffle	waﬄe
strike	ﬅrike

Many fonts have internal pointers to additional glyphs within the font that are to replace certain sequences of letters. The Unicode Standard specifies standard locations for many of these, but some fonts choose to place some or all of them elsewhere.[40] Unfortunately, not all fonts report such information[41] when queried, and not all rendering engines make consistent use of this information.

Such substitutions are handled by the same rendering engines referred to in the description of Figure 4. This is done based on instructions found in a variety of locations throughout the system and, in the case of Ligatures, in the fonts themselves! So how is this applicable to database custodians?

---

38 … in some cases, as alternate spellings, such as encyclopaedia (a & e) or encyclopædia (the æ ligature) for encyclopedia.

39 Just so you know: there is a technical distinction between the terms *Digraph* and *Ligature*, although both are formed by combining two glyphs in a single Character Cell. In a Digraph, the glyphs remain separate but are placed close together as in Figures 7 (ü), 8 (ñ) and 9 (½). In a Ligature such as that in Figure 10 (ﬄ), the individual glyphs are fused into a single glyph. Looking back at Figure 4, the third Character Cell contains a Digraph (ค็); presumably then, the contents of the first Character Cell (นี้) would be a Trigraph but that term, although familiar to C and C++ programmers, is seldom used in this context.

40 If you are interested, "elsewhere" refers to what Unicode calls "Private Use Areas" (PUAs), blocks which don't have and are guaranteed never to have any formal character assignments. The "standard" Unicode Block for ligatures that had no earlier assignment is found at http://unicode.org/charts/PDF/UFB00.pdf and is known as Alphabetic Presentation Forms.

41 Much more information about fonts and their effects on multi-lingual, multi-Script database implementations is provided in "Evaluating Fonts for use in Multi-Lingual Documents" – the fourth Design Note in this series.

Whether a particular Ligature is a proper character with an assigned Unicode value or not, it should only be used as a display-only substitute; its actual component characters should be stored on disk and in memory. As you ponder (and you should) what might happen should any such “display-only” Ligatures end up in your database, the example shown in Figure 11c should ﬅrike you as quite a kerfuffle (stored of course as “kerfuﬄe”)!

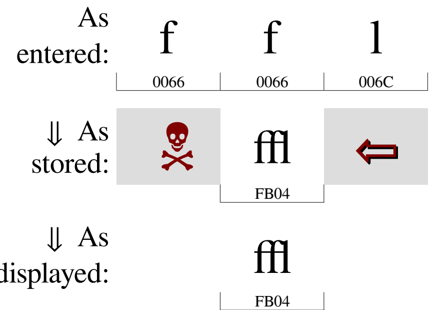


Figure 11c – NO!

Once upon a time, a number of customers, including Ms. Haffter and Mr. Noffliger, could no longer be found in the system when customer service representatives searched by name.

When searching by customer number, however, they appeared. On sorted lists, these folks appeared in unexpected locations, and in some applications their names were displayed with some odd characters.

The lack of ligatures in the names above (and others) began to annoy the marketers: although it isn’t likely that Mr. Noffliger ever noticed, the layout of his name was felt to reflect poorly on the company’s image.

So they began a campaign to edit the names of those clients whose names represented these annoyances. The database, after all, had no constraints on ligatures, and was just as happy to accept ﬄ as ffl.

The professionally-produced promotions on premium paper periodically posted to the company’s clients made the marketing folks in this high-end services business feel justifiably proud[42] of their efforts.

YOUR database soon appeared “haunted” (randomly unreliable) to users outside the marketing group.

The cautionary tale on the left will give you an idea of why even a little exposure to typography can be useful – or alternatively dangerous – even before your organization begins adapting to the world of Unicode and universal databases.

With only the best interests of their company’s image in mind, and with an awareness of typography accrued by experience, members of the team “solved” their problems by manually editing some key client names to make them more aesthetically pleasing in print.

Had they also possessed an equivalent exposure to data management, however, they would have realized that the ﬄ Ligature clearly fell into the category of “formatting” and that storing more than one piece of information in a single data element violates one of the cardinal rules of database design.[43]

Once this issue was identified (by realizing that only one press of an arrow key was required to jump across what should have been three characters), the data could be restored. As data custodians, it is important, however, not to dismiss the marketing department’s concerns (and the spirit) leading to this issue.

Once identified, most issues of this type can be corrected. Working with the marketing department, the commercial printer, and interested representatives from within the Company, a recommended selection of fonts was examined, tested, and agreed upon as the organization’s new “standard” selection. Test documents were created to make it easier to evaluate new fonts to insure they met the criteria established by each participating group before being introduced.

This difficulty with composite characters is one reason the UTC has taken the position that no new digraphs should be encoded[44] in the standard. Other font-related issues can appear even in the simplest of command line database reports:

Editors and terminal emulators used by programmers and DBAs seldom ever use proportionally spaced fonts. For most such uses, variable width fonts are counter productive, and even confusing during development. Since SQL queries tend to pad their output columns with spaces, even when specific formatting commands are given, names (for instance) in differing Scripts such as the surnames McCarthy and โขสเทอ์ may not be properly aligned, resulting in a somewhat unprofessional appearance.

```
> SELECT ID, GIVEN_NAME, SURNAME …

------ ----------- --------- -----
ID     GIVEN_NAME  SURNAME   NEXT
------ ----------- --------- -----
    56 Aphrodite   Smith     Next
    88 Agamemnon   McCarthy  Next
   126 แจข         โขสเทอ์      Next
   128 บีเช์         โขสเทอ์      Next
   245 Aristotle   Murphy    Next
```

42 Almost as proud as we are of the alliteration. This tale, by the way, is simplified from reality, but is based on actual events.

43 This shouldn’t surprise the intended audience – database designers: combining both data elements and their desired formatting in one column is a major source of what the book *Business Database Triage* (see page 18) refers to as “Database Constipation,” and which can stymie many efforts at extending existing schemas to support new business opportunities, and – more often than designers realize or will admit – cause significant reductions in data integrity.

44 See the interesting Q&A on the http://unicode.org/faq/ligature_digraph.html page.

Like most such situations you will encounter, an understanding of why this occurs [45] will lead to a rather quick solution. In this case, replacing the system font was not a good option as it would have had unpredictable results.

Modifying the queries to include `CHR(9)` (a Tab character) between each column name permitted the outputs to be piped into a simple word processor template – not only was the layout problem solved, but a more attractive output was made possible for all the organization's batch reports.

Original

```
SELECT ID
     , GIVEN_NAME
     , SURNAME
FROM PERSON ...
```

Modified with Tabs

```
SELECT ID
  , CHR(9), GIVEN_NAME
  , CHR(9), SURNAME
FROM PERSON ...
```

As astute readers will have realized by now, it is not only the sizing of database columns themselves that is affected by a change to multi-language, multi-script systems, but the sizing of variables in programs, columns in reports, et cetera.

Additionally, the behavior of `ORDER BY` clauses should be understood and set based on your company's specific needs.

FIGURE 12: (page 9) This example is only provided as a teaser of things to come. If the first two Devanagari (the Script) characters of this Hindi (the Language) phrase are entered, someone unfamiliar with the Script might suspect that it must be written from right-to-left. It isn't. The हि character reversal is an example of why the term "Complex Text Layout" began to be used in the first place. This is discussed in more detail in the next Design Note of this series.

Locales

The Locale on a given machine specifies the default Language, Country, Character Encoding (e.g. UTF-8), default sorting orders (Collation), address formatting (placement of postal codes), currency, currency formatting, number formatting (e.g. comma versus period as decimal point), and default paper sizes used by the operating system. Aside from the Character Encoding, the Locale setting places few restrictions on the Data you will be able to enter or store.

Several RDBMS applications (often incorrectly called "databases") have the ability to set a Locale for the database itself separately from the Operating System. Successful implementation of a multi-Language, multi-Script database installation or upgrade requires that these two be compatible, or at least have support for exchanging data transparently.

Typefaces, Glyphs & Fonts

These terms are often used rather indiscriminately, regardless of the sources consulted and, like other terms discussed in this section, aren't necessarily relevant to the larger thrust of this paper. Nonetheless, since they are found throughout, it seemed appropriate to at least comment on them briefly to avoid any misunderstanding.[46] I'll reiterate that these definitions are not intended to be technically complete, and are therefore not necessarily accurate.

Consider the following examples:

A	A	A	A	A
U+0041	U+0041	U+0041	U+0041	U+0041
A	A	A	A	A

Five Identical Latin "A" Characters
with Different Typefaces/Styles and Glyphs

ก	ภ	ถ	ฦ	ฤ
U+0E01	U+0E20	U+0E16	U+0E26	U+0E24
d [47]	4	5	?	A

Five Different and Unrelated Thai Characters
with Identical Typeface and Style

In the table above, five different character 'styles' are shown on the left, including both sans-serif and serif varieties as well as several others; these are all the same character expressed with different symbols or glyphs. These are different typefaces used to render the same alphabet.

---

45 You recognize immediately that your SQL utility fails to properly distinguish between Characters and Character Cells.

46 For those with an interest, one source that we consider both authoritative and entertaining is "The Elements of Typographic Style" by Robert Bringhurst; ISBN: 978-0-88179-212-6, now in its fourth edition.

47 As with earlier examples, these keystrokes are provided to permit easy typing/testing with a Thai IME on a Latin keyboard.

In contrast, five different Thai characters are shown on the right. What look like similar stylistic differences or decorative elements on each of these characters are integral parts that differentiate the character symbols; these are different characters – not different typefaces.

A **Typeface** is a coordinated set of stylistically consistent designs for at least one alphabet and, most likely a supplemental array of ancillary symbols intended for use with whatever languages might use that alphabet. Typefaces intended for the same alphabet will contain the same core alphabetic characters and symbols, but the remainder of the ancillary symbols may not be the same for all typefaces.

A **Glyph** is a single unique drawing or other realization of one symbol in a typeface. Such symbols are not necessarily equivalent to complete characters; some characters may be formed from more than one glyph and some character cells, as we've already seen, may contain more than one glyph and even more than one character.

A **Font** was originally a collection of blocks (such as the "g" shown on the left) in a specific size and style for a particular typeface, but that distinction has become less relevant as computer technology and scalable typefaces became available. A Font now would be more usefully described as a set of mathematical instructions with which a computer can transfer the design of each glyph to some display device.[48] A key difference in the current definition of the term font is that many, if not most, currently used fonts contain alphabets and/or symbols from more than one of the Unicode blocks or Scripts described earlier.

A **Font Family** is a related group of fonts that are stylistically matched, but in different weights or styles; examples would be a set of Bold or Italic versions. While these may be separate font files, most applications list only the standard version and use the others transparently when called for. Where these variants don't exist, many systems will "thicken" the glyphs to simulate a Bold for instance, much like repeatedly over-striking the character on manual typewriters allowed typists to add some emphasis.[49] Similarly, a glyph may be slanted to simulate a missing Italic variant.

Keeping an Open Mind

Before closing this first Design Note, we should mention that one previously unstated aim of this series on dealing with the introduction of multi-lingual, multi-script data into a system is to convince developers that one of the best precursors to success with such a project is your ability to keep an open mind. We'll conclude with a few examples:

ก	ภ		ถ	ฦ		ฤ
l	a	d	b	g	q	p
1	2	3	4	5		6

Latin Script can be just as confusing to those who only know Thai!

For those concerned that the characters in foreign alphabets are far to similar to each other for us to easily recognize the differences, such as the five distinct Thai characters given above (ก, ภ, ถ, ฦ, ฤ), here is a comparison with some Latin characters as a rebuttal of sorts. Remember: Perspective!

Column 1 shows a basic straight uncomplicated character in Thai and Latin Scripts. The confusing Thai character in column 2, with its looping to the left is balanced by two Latin characters which add some loops on their left. And then there are the subtle differences between the Latin "d" and "b" – similar to the differences between the ภ and ถ, but not nearly as confusing as the subtle differences between the Latin "g" and "q" and so forth. Then there are the additional Latin character variants: does this "*a*" really resemble the "a" above? Why would you even suspect they're the same character? Does this "**g**" really resemble the "g" above?

Then consider the consistent (but only to us) manner in which b becomes an upper case B, while d becomes an upper case D. And how does one explain the capitalization of g and q, which become G and Q respectively?

It's all about what you grew up with. Keep an open mind. Avoid making assumptions based on your own language - you know all the clichés about assumptions – as developers we've all been bitten by these at least once – in the world of multi-lingual, multi-script development we need to consciously and deliberately be aware of such assumptions.

48 Which, in this context, also includes printers and similar devices.

49 Much more information about fonts and their effects on multi-lingual, multi-Script database implementations is provided in *Evaluating Fonts for use in Multi-Lingual Documents* – the fourth Design Note in this series.

So, when preparing for the upgrade, you need to – as the cliché goes – make sure to dot your i's and cross your t's.

Or Not: Another example of an Alphabet and Script characteristic that causes some confusion for westerners with little exposure to the world at large is the occasional absence of the aforementioned *Tittle*.[50] In Latin Script we are quite familiar with the idea that a lower case i always has this tittle, and don't even notice that its uppercase counterpart I doesn't. So the first exposure to the Turkish alphabet sometimes comes as a surprise to many. Turkish has two separate letters – i and ı – which have uppercase counterparts I and İ. Certainly that is more consistent than Latin Script![51] The Turkish j and its uppercase J, however, still retain the inconsistency we Westerners consider "normal." (Aside: congratulations if you noticed that the ﬃ ligature replacing the ffi sequence has no tittle on its "i" (or is it actually an "ı"?)! [52]

During the design phase of any upgrade, you need to, as our British counterparts would say, get everything sorted out. Any sort of sort (sorry) is possible; the primary problem you will face is determining what users expect the sort order to be in any context, particularly when multiple scripts are involved in a single list.

Related to sorting is searching: you should examine just how the various applications (including RDBMS products), programming languages and semi-languages (e.g. html, xml, etc.) handle regular expressions for Unicode characters. Unicode character storage (including the variable width UTF-8 format) can make this interesting. In addition to a familiarity with *Exploring UTF-8*, the third of this Design Note series, other sources to study include:

http://www.unicode.org/reports/tr18/tr18-19.html – the Unicode Standard for Regular Expressions

http://www.regular-expressions.info/unicode.html – an excellent group of pages on Regular Expressions

http://www.fileformat.info/info/unicode/category/index.htm – listing of Unicode Character Categories

Conclusion

With this broad introduction to the World's Alphabets and their quirks, the next step is to add more detail by proceeding to the second Design Note in this series: *Exploring Complex Text Layout*, which can be downloaded from www.AntikytheraPubs.com/i18n.htm.

50 See the call out titled "Frivolous Aside" on page 10.
51 The Unicode values for these respectively are 0x0069 (i), 0x0131 (ı), 0x049 (I), and 0x0130 (İ).
52 Ligatures for the two initial letters (ti) and two middle letters (tt) of the word "tittle" as printed here are also occasionally seen as displayed Ligatures with some fonts, but not with the one used here, which only supports "standard" Unicode – sorry.

Appendix – Observing Modifier Key Behavior in More Detail

The behavior of modifier keys such as the shift key is reasonably straightforward. More detailed examples of how key presses are modified to reflect the user's intentions – including the use of virtual keyboards used with Input Method Editors (IMEs), as well as a means for observing these in action, are provided below.

The outputs presented are those displayed using the Linux key press monitor utility `xev`, although there are similar utilities for any commonly used operating system. The sequences described on page 8 are repeated first to show how they are displayed using `xev`. First is the entry of the single letter a:

User Key Press Sequence	Output reported by the Linux xev utility
Press and release the A key. State 0x10 means 'unmodified.' Keycode 38 is the key identifier. Keysym 0x61 is the unmodified code used as the default interpretation of keycode 38. Unicode u+61 is "a". The KeyRelease event forwards the 'a' character (`u+61`) to the next step.	**KeyPress event**, serial 34, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 174984398, (103,-9), root:(990,472), state 0x10, keycode 38 (keysym 0x61, **a**), same_screen YES, XLookupString gives 1 bytes: (61) "a" XmbLookupString gives 1 bytes: (61) "a" XFilterEvent returns: False **KeyRelease event**, serial 37, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 174984541, (103,-9), root:(990,472), state 0x10, keycode 38 (keysym 0x61, **a**), same_screen YES, XLookupString gives 1 bytes: (61) "a" XFilterEvent returns: False

The next example shows how the shift key alters the state of the system so that key presses entered are altered to produce a character other than what an unmodified bit stream from the keyboard would normally produce. In this case, the entry of an uppercase/capital A is shown:

User Key Press Sequence	Output reported by the Linux xev utility
Press the Shift key and hold it until the next key is pressed. Note that the state for the next key press will be 0x11, 'shifted.'	**KeyPress event**, serial 37, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 174986619, (103,-9), root:(990,472), state 0x10, keycode 50 (keysym 0xffe1, **Shift_L**), same_screen YES, XLookupString gives 0 bytes: XmbLookupString gives 0 bytes: XFilterEvent returns: False
Press and release the A key. With the state 'shifted,' keycode 38 is now interpreted as keysym 0x41. Unicode `u+41` is "A". The KeyRelease event forwards the 'A' character (`u+41`) to the next step.	**KeyPress event**, serial 37, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 174987066, (103,-9), root:(990,472), state 0x11, keycode 38 (keysym 0x41, **A**), same_screen YES, XLookupString gives 1 bytes: (41) "A" XmbLookupString gives 1 bytes: (41) "A" XFilterEvent returns: False **KeyRelease event**, serial 37, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 174987295, (103,-9), root:(990,472), state 0x11, keycode 38 (keysym 0x41, **A**), same_screen YES, XLookupString gives 1 bytes: (41) "A" XFilterEvent returns: False
Release the Shift key. Releasing the shift key sets the state for the next key press to 0x10.	**KeyRelease event**, serial 37, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 174987717, (103,-9), root:(990,472), state 0x11, keycode 50 (keysym 0xffe1, **Shift_L**), same_screen YES, XLookupString gives 0 bytes: XFilterEvent returns: False

The third example illustrates how the interpretation of key presses is altered when an Input Method (in this case, iBus) is used and the Thai TIS-820 keyboard layout is activated as described in Step 13. The actual switch to an alternate

keyboard mapping also produces output from the xev utility (as would be expected, since xev responds to mouse as well as keyboard actions), but these are ignored here. The next two examples use the same physical keys as above:

User Key Press Sequence	Output reported by the Linux xev utility
Press and release the A key. State 0x10 means 'unmodified.' Keysym 0xdbf is the unshifted code altered by the input method to replace the default interpretation of keycode 38 with the UTF-8 sequence e0b89f. Extracting the Unicode value from the UTF-8 sequence gives u+0E1F. The KeyRelease event forwards the character 'ฟ' (u+0e1f) to the next step.	**KeyPress event**, serial 43, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 175013665, (840,-344), root:(1727,137), state 0x10, keycode 38 (keysym 0xdbf, **Thai_fofan**), same_screen YES, XLookupString gives 3 bytes: (e0 b8 9f) "ฟ" XmbLookupString gives 3 bytes: (e0 b8 9f) "ฟ" XFilterEvent returns: False **KeyRelease event**, serial 43, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 175013808, (840,-344), root:(1727,137), state 0x10, keycode 38 (keysym 0xdbf, **Thai_fofan**), same_screen YES, XLookupString gives 3 bytes: (e0 b8 9f) "ฟ" XFilterEvent returns: False

Thus, pressing the key marked with an A produces the Thai character ฟ.

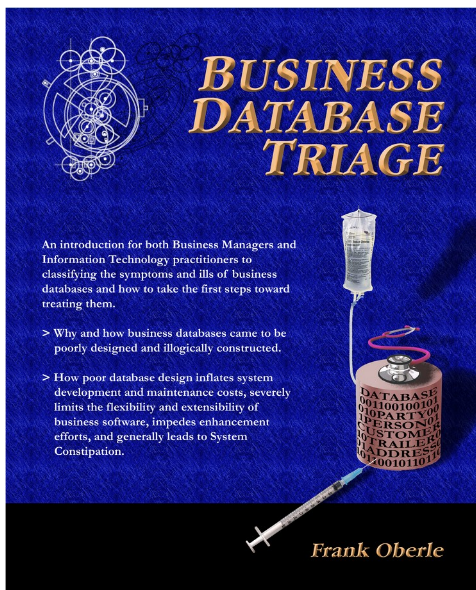
The final example shows how the shift key produces an entirely different Thai character from the same key; as mentioned earlier, Thai, like many scripts, has no concept of "capital" letters.

User Key Press Sequence	Output reported by the Linux xev utility
Press the Shift key and hold it until the next key has been pressed. Note that the state for the next key press will be 0x11, 'shifted.'	**KeyPress event**, serial 43, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 175015953, (840,-344), root:(1727,137), state 0x10, keycode 50 (keysym 0xffe1, **Shift_L**), same_screen YES, XLookupString gives 0 bytes: XmbLookupString gives 0 bytes: XFilterEvent returns: False
Press and release the A key. Keysym 0xdc4 is the shifted code altered by the input method to replace the default interpretation of keycode 38 with the UTF-8 sequence e0b8a4. Extracting the Unicode value from the UTF-8 sequence gives u+0E24. The KeyRelease event forwards the character 'ฤ' (u+0E24) to the next step.	**KeyPress event**, serial 43, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 175016495, (840,-344), root:(1727,137), state 0x11, keycode 38 (keysym 0xdc4, **Thai_ru**), same_screen YES, XLookupString gives 3 bytes: (e0 b8 a4) "ฤ" XmbLookupString gives 3 bytes: (e0 b8 a4) "ฤ" XFilterEvent returns: False **KeyRelease event**, serial 43, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 175016645, (840,-344), root:(1727,137), state 0x11, keycode 38 (keysym 0xdc4, **Thai_ru**), same_screen YES, XLookupString gives 3 bytes: (e0 b8 a4) "ฤ" XFilterEvent returns: False
Release the Shift key. Releasing the shift key sets the state for the next key press to 0x10.	**KeyRelease event**, serial 43, synthetic NO, window 0x3c00001, root 0x2c5, subw 0x0, time 175017291, (840,-344), root:(1727,137), state 0x11, keycode 50 (keysym 0xffe1, **Shift_L**), same_screen YES, XLookupString gives 0 bytes: XFilterEvent returns: False

This time, pressing the A key along with either Shift key produces the Thai character ฤ.

Other Publications

Antikythera Publications



www.AntikytheraPubs.com

In addition to an ongoing series of Database Design Notes, Antikythera Publications recently released the book "*Business Database Triage*" (ISBN-10: 0615916937) that demonstrates how commonly encountered business database designs often cause significant, although largely unrecognized, difficulties with the development and maintenance of application software. Examples in the book illustrate how some typical database designs impede the ability of software developers to respond to new business opportunities – a key requirement of most businesses.

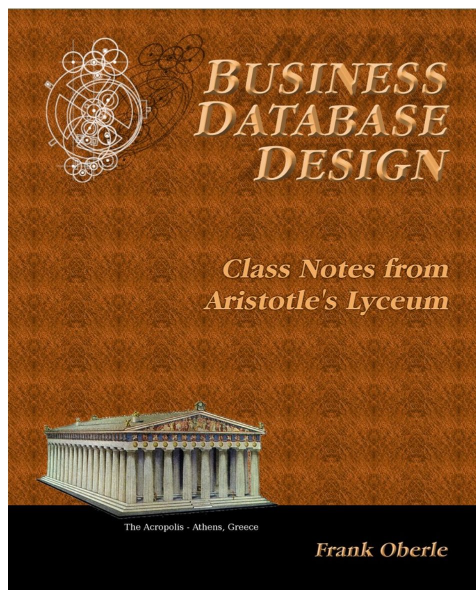
A number of examples of solutions to curing business system constipation are presented. Urban legends, such as the so-called object-relational impedance mismatch, are debunked – shown to be based mostly on illogical database (and sometimes object) designs.

"*Business Database Triage*" is available through major book retailers in most countries, or from the following on-line vendors, each of which has a full description of the book on their site:

CreateSpace: https://www.createspace.com/4513537

Amazon:

www.amazon.com/Business-Database-Triage-Frank-Oberle/dp/0615916937



A follow-up book, "*Business Database Design – Class Notes from Aristotle's Lyceum*" is due to be available in the latter part of 2014.

"*Business Database Design*" leads the reader through the logical design and analysis techniques of data organization in more detail than the earlier work – which concentrated more on understanding and identifying problems caused by illogical database design rather than their solutions.

These logical approaches to data organization, espoused by Aristotle and an "A-List" of his successors, have formed the basis for scientific discovery over more than 2,400 years, and directly led to the technology we deal with today, notably including both relational and object theory.

"*Business Database Triage*" explained the reasons why these principles were virtually impossible to apply during the early years of our transition to the use of computers in business, but since the technology is now sufficiently mature that such compromises can no longer be justified, the time has come to relearn logical data organization techniques and apply them to our businesses.