Regular Expression Matching for Multiscript Databases

Sudeshna Sarkar
Computer Science & Engineering Department,
IIT Kharagpur
email: sudeshna@cse.iitkgp.ernet.in

Abstract

Modern database systems mostly support representation and retrieval of data belonging to different scripts and different languages. But the database functions are mostly designed or optimized with respect to the Roman script and English. Most database querying languages include support for regular expression matching. However the matching units are designed for the Roman script, and do not satisfy the natural requirements of all other scripts. In this paper, we discuss the different scripts and languages in use in the world, and recommend the type of regular expression support that will suit the needs for all these scripts. We also discuss crosslingual match operators and matching with respect to linguistic units.

1 Introduction

Language is a defining feature of human civilization, and many languages and scripts have come into existence that are used by people around the world. As multilingual requirements are coming to the forefront in today's world, databases are required to effectively and efficiently support storage, indexing, querying and retrieval of multilingual texts. Various databases now offer some multilingual support, but most of the features are still designed and optimized for the Roman script.

Database languages like SQL use certain string matching primitives. However these primitives are very rudimentary and have been primarily designed with the Roman alphabet in view. But when one considers the diversity of writing systems, firstly there is a need to re-think these primitives for monolingual matching, and evolve patterns that accommodate the diverse requirements for the different scripts. Secondly, we also need to address the requirements of multilingual databases, where information from different languages and scripts are stored in the same database. For multi-script databases, in addition to being able to handle matching requirements in various individual languages, we need to have cross-lingual regular expression matching operators. Thirdly, as text data is becoming increasingly common, databases are being used to store running text data. The use of special matching units in addition to the letter-based matching methods will extend the power and functionality of databases with respect to text data.

In this work we will inspect some of the important types of writing systems, and discuss effective operators keeping in view the requirements of these writing systems. The most popular pattern matching operator in SQL and other popular database languages is LIKE which supports wildcard characters for matching a single

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

character or a sequence of characters. We find that in some writing systems the elementary character units are different from a visual graphemic unit, and the notion of what a character stands for varies across different writing systems. We discuss this diversity, and recommend appropriate matching units for various types of scripts and languages. In this work, we propose different standard character classes which need to be defined for the various languages and script families. The support for this may be built into the database. If not it is possible to specify these using user defined functions. We also propose the introduction of a cross-lingual pattern matching operator, so that pattern matching can be supported across different languages. Further, we recommend that databases provide support to linguistic matching units.

The paper is organized as follows: In Section 2, we discuss briefly about the major types of writing systems, in order to understand the meaningful units with respect to the different systems. In Section 3 we discuss the regular expression support in standard databases including SQL. In Section 4, we make our recommendations about defining character classes to better support regular expressions across different scripts and languages. We also introduce a crosslingual matching operator LexLIKE.

2 Writing systems

A *writing system* [2, 11] is a type of symbolic system used to represent elements or statements expressible in language. All writing systems possess a set of base units, which are called *graphemes*, and collectively they form a *script*. Writing systems differ in many ways including the basic units they use and the direction in which the script is written. We now discuss in brief the major classes of writing systems.

2.1 Types of writing systems

Abjads Abjads [3] which are used by scripts of languages like Arabic, Hebrew and Persian were the first type of alphabet to be developed. These scripts represent only consonants, and not vowels in the basic graphemes. They have one symbol per consonantal sound. In rare cases, the vowel marks may optionally be included by means of diacritics. Most of Abjads are written from right to left.

In the Arabic script, words are written in horizontal lines from right to left, but numerals are written from left to right. The long vowels /aː/, /iː/ and /uː/ are represented by letters. Vowel diacritics, which are used to mark short vowels, and other special symbols appear only in a few religious and children texts.

Alphabets An alphabetic writing system has a set of letters, or graphemes, each of which roughly represents one or more phonemes, both consonants and vowels, of the spoken language. Many languages (e.g., those based on the Latin, Cyrillic, Greek and Armenian alphabets) use multiple letter cases in their written form (e.g., English uses majuscule or upper case and minuscule or lower case). The most widely used alphabets are the Latin or Roman alphabet and the Cyrillic alphabet, which have been adapted to write numerous languages.

In some cases combinations of letters are used to represent single phonemes, as in the English 'ch'. A pair of letters used to write one sound or a sequence of sounds that does not correspond to the written letters combined is called a *digraph*. In some languages, digraphs and trigraphs are counted as distinct letters in themselves, and assigned to a specific place in the alphabet, separate from that of the sequence of characters which composes them, in orthography or collation. For example, the Dutch alphabet includes 'ij' which is usually collated as a single unit. The Croatian alphabet includes some digraphs such as 'dž', 'lj' and 'nj' which are treated as single letters. Other languages, such as English, split digraphs into their constituent letters for collation purposes. Some alphabetic writing systems use extra letters and ligatures. A ligature occurs where two or more letter-forms are joined as a single glyph. French has the ligatures œ and æ. The German 'esszett' represented as ß is a ligature that is replaced by SS in capitalized spelling and alphabetic ordering. In English, æ is treated as a spelling variant and not an independent letter. But languages like Icelandic, Danish and Norwegian, treat æ as

a distinct vowel. Many alphabetic systems add a variety of accents to the basic letters, These accented letters can have a number of different functions, such as, modifying the pronunciation of a letter, indicating where the stress should fall in a word, indicating pitch or intonation of a word or syllable, and indicating vowel length. We discuss more about accents in Section 2.2.

Syllabic Alphabets or Abugidas A syllabic alphabet (also referred to as Abugida or alphasyllabary) is an alphabetic writing system which has symbols for both consonants and vowels. Consonants have an inherent vowel. A vowel other than the inherent vowel is indicated by diacritical marks. The dependent vowels may appear above or below a consonant letter, to the left of or to the right of the consonant letter, and in some cases it may have more than one glyph component that surround the consonant letter. A consonant with either an inherent or marked vowel is called an *akshara*. Vowels can also be written with separate letters when they occur at the beginning of a word or on their own. The vast majority of Abugidas are found in South and Southeast Asia and belong historically to the Brahmi family.

Devanagari (used to write Hindi, Sanskrit, Marathi, Nepali, and some other North Indian languages), Bengali, Gurmukhi script, Tamil, Telugu, Tibetan, Burmese, Thai are some examples of Abugidas. The inherent vowel in Devanagari is /a/. Thus the first consonant in Devanagari क stands for 'ka' and not just the consonant sound 'k'. 'ki' is written by using a diacritic along with 'ka' that stands for the vowel sound 'i'. The same diacritic will be used to get 'ti' from 'ta'. Diacritics are also used for nasalizing the vowel in the syllable (e.g., किँ which represents 'ki' nasalized.). A lonely consonant or dead consonant without any following vowel is sometimes indicated by a diacritic e.g. halant or virama in Devanagari (क्). In a few cases, a separate symbol is used (e.g., ৎ in Bengali is the dead form of the consonant ত When two or more consonants occur together, special conjunct symbols are often used which add the essential parts of first letter or letters in the sequence to the final letter. Such consonants are called half consonants. The final character may retain some similarity with the combining consonants or can be different. Some examples from Devanagari are न् + त = न्त, क् + ष = क्ष. The half forms do not have an inherent vowel. A consonant cluster can be composed of partial parts of the constituent consonants or can have a completely different form. Thus the effective orthographic unit in Abugidas is a syllable consisting of *C*V* (*C* stands for a consonant, and *V* for a vowel) in general.

Syllabaries A syllabary is a phonetic writing system consisting of symbols representing syllables. A syllable is often made up of a consonant plus a vowel or a single vowel. Some examples are hiragana (Japanese), katakana (Japanese), Inuktitut, Cherokee (Tsalagi). Some writing systems like Iberian use a script that is a combination of syllabic and alphabetic. Modern Japanese is written with a mixture of hiragana and katakana, plus kanji.

Logographic writing systems In logographic writing systems, each character or logograph represents a morpheme which is the minimal unit in a language that carries meaning. A logogram may represent a word, or part of a word like a suffix to denote a plural noun. Many Chinese characters are classified as logograms. But no logographic script is comprised solely of logograms. All contain graphemes which represent phonetic elements as well. Each Chinese character represents a syllable and also has a meaning. The phonetic elements may be also used on their own.

2.2 Diacritics

A diacritic is a small sign added to a letter. As we have already pointed out, several types of diacritic [9, 10] markers are used in different scripts. It often alters the phonetic value of the letter to which it is added, but in some cases it may modify the pronunciation of a whole word or syllable, like the tone marks of tonal languages. As we have already discussed, Abugidas and, in some cases, Abjads (such as Hebrew and Arabic script) use diacritics for denoting vowels. Hebrew and Arabic also indicate consonant doubling and change with diacritics.

Diacritics are used in Hebrew and Devanagari for foreign sounds. As noted earlier, Devanagari and related Abugidas use a diacritical mark called a halant to mark the absence of a vowel. The Japanese hiragana and katakana syllabaries use the dakuten and handakuten symbols to indicate voiced consonants.

A letter which has been modified by a diacritic may be treated as a new, individual letter, or simply as a letter-diacritic combination, in orthography and collation.

Some of these languages use diacritics that are considered separate letters in the alphabet, a few of which are mentioned below. Estonian has the following distinct letters õ, ä, ö, ü which have their own place in the alphabet, between w and x. Finnish uses dotted vowels ä and ö which are regarded as individual letters. Hungarian uses the acute and double acute accents. In Spanish, the character ñ is considered a letter, and is collated between n and o. Icelandic uses acute accents, digraphs, and other special letters, which each have have their own place in the alphabet.

On the other hand, some languages with diacritics do not produce new letters. For example, French uses *grave*, *acute*, *circumflex*, *cedilla* and *diæresis*. German uses the *umlauted* vowels ä, ö or ü to indicate vowel modification. Thai has its own system of diacritics derived from Indic numerals, which denote different tones.

2.2.1 Tones

Tone is the use of pitch in language to distinguish words. Many of the languages of South-East Asia and Africa are tone languages and the script contains marks to indicate tones. The tone contour represents how the pitch varies over a syllable for a tone in a tonal language. It is usually denoted by a string of two or three numbers, or an equivalent pictogram. Many languages have tones and there are several schemes for representing tones in the orthography of the language: tone letters, tone diacritics, and superscript numbers. In African languages, usually a set of accent marks are used to mark tone. The most common accent marks are: high tone denoted by acute (as in $\acute{a}$), mid tone denoted by macron (as in $\bar{a}$) and low tone denoted by grave (as in $\grave{a}$. Several variations are common, and often one of the tones is omitted. Omotic languages have a more complex tonal system which are indicated by numbers from 1 to 5. Contour tones are indicated by a pair of numbers e.g., 14, 21.

In Thai (an Abugida) and Vietnamese (an alphabet) tones are indicated with diacritics. In a few cases, a script may have separate letters for tones, as is the case for Hmong and Zhuang. Numerical systems to denote tones are more common in Asian languages like Chinese.

2.3 Word segmentation

Apart from differing scripts, writing systems also differ in other dimensions. In many written languages, the word boundary is explicit. For example, many languages like English and Hindi have spaces marking word boundaries. Arabic uses different letter shapes for word initial, medial and final letters. But word segmentation is a problem in several Asian languages that have no explicit word boundary delimiter, e.g. Chinese, Japanese, Korean and Thai. In Sanskrit, words are joined together by *external sandhi* which involves the process of coalescence of the final letter of a word with the initial letter of the following word.

2.4 Writing systems on the computer

Writing systems on the computer make use of a character encoding for a language which assigns a unique value to represent each of the characters in that language. Different ISO/IEC standards are defined to deal with each individual writing system to implement them in computers (or in electronic form). There are several different encodings like ASCII and the ISO Latin family of encodings which use one byte to represent a character. ISO 10646 is an international standard, by ISO and IEC. It defines UCS, Universal Character Set, which is a very large and growing character repertoire, and a character code for it. Since 1991, the Unicode [1] Consortium has worked with ISO to develop The Unicode Standard ("Unicode") and ISO/IEC 10646 in tandem. Unicode adds

rules and constraints such as rules for collation, normalization of forms, and so on. In Unicode, each character, from the writing systems of all languages, is given a unique identification number, known as its *code point*.

While Unicode is a standard for representation of various languages, if one considers the Unicode representation, in many cases there is a difference between a grapheme or code represented in the computer and a visual unit or glyph. This is especially evident in the case of Abugidas, where three different approaches are used in Unicode representation. Regular expression support for a script must take into account the model of representation used for that script.

The Devanagari model, used for most Indian languages, represents text in its logical order and encodes an explicit *halant* (*virama*) symbol. For example, the syllable 'ki' is written in Devanagari by placing the diacritic for the vowel 'i' to the left of 'ka' as in कि, but is represented in Unicode as the code for 'ka' followed by the code for 'i'. The diacritic marker for the vowel 'o' in Bengali surrounds the consonant letter to which it is attached: ক ('ka') with vowel 'o' is represented as 'ko' as in কো, but the Unicode representation of 'ko' is U+0995 U+09CB representing 'ka' and 'o' respectively.

The Thai model represents text in its visual display order. For example, consider the syllable 'ke'. In Thai script, the diacritic for the vowel 'e' comes to the left of the consonant symbol 'ko kai' (ก) as in เก. In Unicode this syllable is represented by U+0E40 U+0E01, where the code for the vowel comes before the code for the consonant, but 'kae' as in แกะ is written as U+0E41 U+0E01 U+0E30, where the vowel marker is indicated by two codes, one coming before and one after the code for the consonant.

The Tibetan script is syllabic with consonant clusters written using groups of characters, some of which are stacked vertically into conjunct characters. Vowels (other than the inherent /a/) are indicated using diacritics. Morphemes are separated by a dot (tsheg). The Unicode representation of the Tibetan script [1] uses one set of codepoints which are used to represent either a stand-alone consonant or a consonant in the head position of a vertical stack, and a second set of codepoints to represent consonants in the subjoined stack position. The code for the vowel sign is always placed after the consonants in the Unicode representation. Thus a syllable is represented by a consonant symbol followed by zero or more subjoined consonant symbols followed optionally by vowel diacritics. The syllable སྒུ, is represented by U+0F66 U+DF90 U+0FB2 U+0F74 where the four codes stand for the following four characters, ས◌ྒ◌ྲ◌ུ, the first being a head consonant('sa'), the next two subjoined consonants ('ga' and 'ra'), and the third a vowel diacritic ('u').

3 Regular expression and multilingual support in databases

We will now turn our attention to reviewing the regular expression support in standard database systems, and then discuss their suitability for multilingual handling.

3.1 Multilingual datatype and collation support

Most databases now have support for storing multilingual characters in fixed and variable length formats, and also have support for representation in Unicode. Further, since the lexicographical order may be different in different scripts, the SQL standard allows one to specify collation sequences to index the data, as well as to correctly sort the data for display order. The database server specifies a default collation sequence to be specified during installation. Most database servers allow collation sequences to be specified at the table and individual column level while creating a database. Collation can be changed using alter database. Database systems like Oracle 10g [12], SQL Server 2005 define different collation sequences for many languages including ones that are case insensitive and accent insensitive. Similar to sort order, match can be case insensitive or diacritic insensitive.

^	Matches start of line
$	Matches end of line
?	Matches zero or one occurrence of the preceding element
*	Matches zero or more occurrences of the preceding element
{n}	Exactly n repetitions of the preceding element
[abc]	Character list, matches a, b or c
[b-g]	Matches b, c, d, e, f or g
[.ce.]	Matches one collation element including digraphs e.g., [.ch.] in Spanish.
[:cc:]	Matches character classes e.g., [:alpha:] matches any character in the alphabet class. Some of the defined character classes are [:alnum:] for all alphanumeric characters, [:digit:] for all numeric digits, [:lower:] for all lowercase alphabetic characters, etc.
[=ec=]	Matches equivalence classes. For example [=a=] matches all characters having base letter 'a'. A base letter and all its accented variations constitute an equivalence class.

Table 1: Additional wildcards supported in Oracle

3.2 Regular expression support in databases

Regular expressions are a powerful tool for matching text data. A regular expression comprises one or more character literals and/or metacharacters.

SQL SQL uses the LIKE operator for regular expressions. This allows one to match literals as well as patterns of a single character or multiple characters. The usage of LIKE is as follows:

select "column_name" **from** "table_name" **where** "column_name" **like** PATTERN

PATTERN can include the following wildcard characters: _ which matches a single character, and % which matches a string of 0 or more characters.

Oracle Oracle SQL [8] has introduced several operators including regexp_like and regexp_substr. Oracle's implementation of regular expressions is based on the Unicode Regular Expression Guidelines and meant to encompass a wide variety of languages. regexp_like has the following syntax: regexp_like (string, pattern, [parameters]);. The optional argument parameters can have different values including i (to match case insensitively) and c to match case sensitively. In addition to the _ and % wildcard characters which are used in like, regexp_like uses regular POSIX and Unicode expression matching. Some of the wildcards used are shown in Table 1. However though there is support for multiple scripts, all scripts are treated uniformly.

3.3 Crosslingual matching

In multilingual databases there is very little support for comparison of strings across different scripts. Equality comparison of strings from different languages can be useful for proper nouns and can be done to strings from an equivalent set of languages under equivalence of individual character sets. Kumaran and Haritsa [5,6] proposed a new SQL operator called LexEQUAL for phonetic matching of specific types of attribute data across languages. It is suggested that the LexEQUAL operator be implemented based on parameterized approximate matching in phoneme space. It requires a G2P (Grapheme to Phoneme) for the concerned languages.

select BookTitle, Author **from** Books **where** BookTitle **LexEQUAL** 'Himalaya'
InLanguages English, Arabic, Marathi, Bengali

will return entries where the BookTitle column has strings that are phonetically close to 'Himalaya' and can be in one of the languages specified. The SemEQUAL operator [7] has been proposed by Kumaran and Haritsa for semantic matching of multilingual attribute data. The query

select BookTitle, Author **from** Books **where** Category **SemEQUAL** 'History'
InLanguages English, Hindi, French

will return entries where category is 'History' for English books, 'Histoire' for French books, इतिहास for Hindi books. A variation, 'SemEQUAL ALL' is used to retrieve additional entries whose category column contains synonyms of 'History' in a thesaurus.

3.4 Lexical Query processing

In addition to these, with the growing popularity of text processing, the database engines need to build in support for functional units of words. Microsoft SQL Server 2005 has good support for full-text search (SQL FTS) [4]. This allows efficient querying with large amounts of unstructured data. In contrast to the LIKE predicate, which only works on character patterns, full-text queries perform linguistic searches against this data, by operating on words and phrases based on rules of a particular language. Full Text Searching allows for string comparisons, returning both results and a matching score or weight. Typically, full text index may be enabled on some selected columns in a database table. Four T-SQL predicates are involved in full-text searching: freetext, freetexttable, contains, and containstable. The query

select BookTitle **from** Books **where freetext**(BookTitle, 'database')

finds a word having 'database' as stem anywhere in the BookTitle column. It matches with "Database Systems" as well as with "Principles of Databases". freetexttable works similarly, but returns its results in a Table object. contains and containstable offer a much more complex syntax for using a full-text indexed column. Some of the capabilities include search for a given prefix term, and searching for different generations of a word using the formsof term. fomsof accepts two arguments inflectional or thesaurus. The inflectional argument causes match with entries containing the same linguistic stem as each word in the search phrase. The thesaurus argument enables a thesaurus based expansion on the search phrase. For each supported language, there will have to exist a single thesaurus file. For example, the following query

select BookTitle, Author **from** Books **where contains**(BookTitle, '**formsof** (inflectional, design)')

retrieved those entries where the BookTitle field contains a word which is any inflected form of 'design'.

contains(*,'**formsof**(inflectional, climb) and **formsof**(thesaurus, mountain)')

will find documents containing inflectional forms of 'climb' and all words meaning the same as 'mountain' (from thesaurus support). During indexing, language options specify how words or tokens are broken from the text stream and stored in the index. At query time, the search phrases supplied in a query are expanded by the parser before searching the full-text index. This expansion process is performed by a language-specific component called a stemmer and the exact expansions that occur will depend on the language-specific rules that have been used.

3.5 Setting up globalization support environment

Databases that support multiple scripts and languages need to provide a support environment for specifying information about the supported languages. As an example, we discuss the support provided in Oracle for setting up locale and language specific support and configuration.

National language support (NLS) in Oracle provides the ability to choose a national language and store, process and retrieve data in native languages. It also ensures that database utilities, error messages, sort order, and date, time, etc. automatically adapt to any native language and locale. Oracle's globalization support is implemented with the Oracle NLS Runtime Library (NLSRTL). The NLSRTL has language-independent functions that allow proper text and character processing and language convention manipulations. The behavior of these functions for a specific language is governed by a set of locale-specific data that is identified and loaded at runtime.

The locale-specific data is structured as independent sets of data for each locale that Oracle supports. The

locale data specifies language, territory, character set, and linguistic sort and can be customized by the user. It is used to define whether matching is to be case or accent insensitive, the collation sequences, the collating elements, etc. Oracle also supports user-defined characters and customized linguistic rules.

NLS parameters in Oracle can be specified by initialization parameters on the server which override the default. It can also be specified as environment variables on the client, with the 'alter session' statement, and in SQL function in reverse order of priority.

4 Recommendations for multilingual and multiscript support

4.1 Units in different writing systems

When we study the adequacy of the regular expressions to handle scripts in multiple languages we note that the notion of what comprises a character varies across languages. Different writing systems use different graphemic units for representation. A single unit may represent a single consonant, a single vowel, a syllable, a consonant and vowel combination, a letter with accent, or a ligature. Different languages have different semantics associated with a character.

1. In Abjads, a grapheme corresponds to a single consonant. A consonant is the main unit along with a few long vowels. The vowel sounds are mostly left out, but in a few cases, they are indicated by diacritics associated with the consonant unit.

2. In alphabetic writing systems, a grapheme corresponds to a single vowel or a single consonant. Consonants and vowels have equal status as letters. Some scripts contain ligatures, some contain digraphs or trigraphs as collating units. Accents or diacritics combine with letters and can be treated as an accented base unit in some cases, and as separate letters in others. In the former case, a character along with its diacritic may be considered as a single unit, or one may consider them as two different units. Ligatures and multi-character collating units can also be treated as one or two units.

3. In Abugidas, a grapheme corresponds to a phoneme unit which represents a consonant vowel combination, or isolated vowel sounds. A consonant letter by itself has an inherent vowel associated with it, usually /a/, which is overridden by diacritic marks for other dependent vowels. In some cases, dead consonants which are not associated with any vowel, and do not combine with the next consonant letter, are used. Dead consonants are usually marked by the diacritic for halant in Devanagari, Bengali, etc. Even though the consonant vowel combination is visually a single unit, this unit is obtained by composing the consonant symbol with diacritic marks. Thus these units are decomposable. Some of the Abugidas also use consonant clusters which are represented by ligatures. Orthographically such ligatures are a single unit, even though they are formed by a sequence of consonants. Half consonants are part of a consonant cluster. For some of these languages, there are four types of units: consonant only (without the inherent vowel), independent vowel, consonant(s) plus vowel unit forming a syllable, and diacritic marker/vowel marker.

4. In syllabic writing systems, a grapheme corresponds to a syllable.

5. In logographic writing systems, a grapheme corresponds to a morpheme. However in most logographic scripts many of the characters have associated phonemes representing syllables.

6. In all languages, the word as a lexical unit plays a very important role. Morphological analysis of a word lets us find the root form which is also called the stem or lemma, and the inflectional affixes.

Based on the above discussion, we identify the following units which are meaningful for different scripts: collating unit, vowel, vowel mark diacritic, consonant, consonant cluster, ligature, single live consonant, dead

consonant, half consonant, syllable, morpheme, diacritic, accent diacritic, tone diacritic and tonal symbols. At the lexical level, the important units are word, stem, suffix, and prefix.

A subset of these units are typically meaningful for a given script and language. These more natural units can be defined as the default units for those scripts. Table 2 shows the meaningful units for the different writing systems, and Table 3 shows the composition of some of the character classes for the different types of scripts.

In addition, when we have a multilingual database where different entries may be in different languages, it is important to define crosslingual matching capabilities. The proposed crosslingual operator LexEQUAL discussed earlier does phonological mappings across multiple scripts. Similar to the LexEQUAL operator, it is also useful to design crosslingual pattern matching operators.

4.2 Recommended character classes for monolingual match

On the basis of the units found in different languages, we identify a few symbols that stand for a single instance of a given unit. We use the character class notion used in Oracle to represent the different units. The SQL wildcard "_" matches a single character. We equivalently introduce the character class [:char:], and use either of them to denote a single character unit. The interpretation of a character in different writing systems is recommended to be as follows:

Abjads:	a vowel or a consonant with optional diacritic marks
Alphabets:	a single consonant or vowel letter or a ligature or a multi-character collating unit or a letter with the associated diacritics
Abugidas:	independent vowel or diacritic vowel or consonant or consonant cluster with or without vowel diacritic
syllabaries:	a syllable
logograms:	a morpheme or a phonetic syllable

Sequences of characters are represented in the following ways:

`%` or `[:char:]*`	is taken to be a sequence of zero or more character units.
`[:char:]+`	is taken to be a sequence of one or more character units.
`[:char:]?`	is taken to be zero or one character unit.

In addition to these, we propose the following more specific wildcards:

`[:c:]`	is used to represent a consonant or a consonant cluster, or a collating unit consisting of two or more consonants.
`[:v:]`	is used to represent a vowel or a ligature of vowels in case of Abjads and alphabets, and an independent vowel in case of Abugidas.
`[:hc:]`	is used to represent a half consonant for some Abugidas.
`[:dc:]`	is used to represent a dead consonant for some Abugidas.
`[:syll:]`	is used to represent a syllable. This is meaningful for syllabaries, is used for independent vowels or consonant clusters in case of Abjads and Abugidas, and for phonetic units in logographs.

The following classes are recommended for diacritic symbols:

`[:dia:]`	is used to represent any diacritic symbol, or a set of diacritic marks.
`[:acc:]`	is used to represent an accent diacritic.
`[:dvow:]`	is used to represent a vowel diacritic.
`[:tone:]`	is used to represent a diacritic representing a tone. This is also used to represent a separate tone symbol as well as numeric tone representations.

The following lexical units are also recommended. These units are meaningful in almost all languages, but it is especially useful to identify these entities in languages where the written language does not use word boundaries,

	Abjads	Alphabets	Abugidas	Syllabaries	Logograms
Character Units	consonant, independent vowel, consonant with dia.	consonant, vowel, digraph, accented unit, ligature	consonant, vowel, syllable, vowel dia. cons cluster	syllable	morpheme
independent vowel	long vowels only	yes	yes		
vowel diacritic	yes(rare)		yes		
ligature	yes	yes		yes	yes
consonant	yes	yes	yes		
consonant cluster			yes (dn)		
live consonant dead consonant half consonant			yes (dn) yes (dn) yes (dn)		
syllable	yes		yes	yes	yes
diacritic	yes	yes	yes	yes	
accent dia.		yes			
tone dia.		yes	yes(Thai)		
morpheme	yes	yes	yes	yes	yes

Table 2: Table showing the meaningful units in different writing systems (dn denotes Devanagari)

Abjads	consonant	=	consonant + consonant with vowel dia
	vowel	=	independent vowel
	syllable	=	consonant with optional diacritic + independent vowel
	char	=	syllable
Alphabets	char	=	consonant + vowel + digraph + accented unit + ligature
Abugidas	consonant	=	live consonant + dead consonant + half consonant
	consonant cluster	=	live consonant + half consonant* live consonant
	syllable	=	independent vowel + consonant cluster with optional diacritic
	vowel	=	= independent vowel + vowel dia.
	diacritic	=	vowel dia + tone dia
	char	=	vowel + consonant + consonant cluster + syllable
Syllabary	char	=	syllable
Logogram	char	=	morpheme

Table 3: Rules showing relations between the different units in different writing systems.

as in Thai, Chinese and Japanese. Identifying these units would require a morphological analyzer or a stemmer for the languages, as well as a text segmentation module for languages with no delimiters.

`[:morph:]`	is used to represent a morpheme. This is the basic unit for logographs. But such units are meaningful for all languages.
`[:word:]`	matches a single word
`[:stem:]`	matches the stem of a word
`[:suffix:]`	matches a suffix
`[:prefix:]`	matches a prefix

Apart from the character classes we define the following matching units:

`[=a=]`	matches all letters with the base character 'a' and includes all diacritics of 'a', as well as upper and lower cases of 'a'. It also matches multiple diacritics on a.
`[=a=:v]`	matches all letters with the base character 'a' with optionally vowel diacritics.
`[=a=:acc]`	matches all letters with the base character 'a' with optionally accent diacritics.
`[=a=:tone]`	matches all letters with the base character 'a' with optionally tone diacritics. Tones may also occur as numeric superscripts or as separate tone symbols. In the case where they are separate symbols, we may use [:tone:].

Note that in order to do these matches, the database system has to know not only the code points which correspond to diacritics like accents, vowel markers, etc., but also there will be language specific differences about the position of the diacritic in the text representation with respect to the base letter.

4.3 Monolingual match operators

Most database systems allow the specification of a collation sequence. For example, by appropriately setting the value of nls_sort in Oracle one can do accent or case insensitive matching. PostgreSQL additionally uses a special keyword ILIKE to make the match case insensitive.

We recognize that various types of case and accent insensitive matching are often very useful, and we recommend that apart from being able to specify the matching scheme, databases should have distinct operators for various levels of matching: literal matching, case insensitive matching, diacritic insensitive matching, as well as phonetic matching. Phonetic matching can also be done across languages and is discussed in Section 4.4.

x LIKE y : x matches y according to the current match environment

x ILIKE y : x matches y insensitive to case

x DILIKE y : x matches y insensitive to diacritics

The semantics for these operators must be specified in the locale for the language. Some examples for successful matches in English are Book ILIKE book and naive DILIKE naïve. For languages like Hindi and Bengali, diacritic insensitive matching should match on the consonants only. Some examples of successful matches might be हिरन DILIKE हरिनी, and कर्म DILIKE करम. While defining the match levels we have to take into account the characteristics of the script. For example, in scripts where the accented forms are treated as different characters, accent insensitive matching with respect to these characters is not common.

4.4 Crosslingual operators

The LexEQUAL operator [5] has been proposed for crosslingual matching. We wish to extend crosslingual matching to pattern matching. Our proposed operator LexLIKE is modeled based on LexEQUAL, and can be used for phonological matching between two strings or between a string and a pattern belonging to one language or two different languages. In case of crosslingual match, the user has to specify the set of target languages, or * to specify any language, and an optional Threshold parameter can help tune the degree of match between the string and the pattern. The syntax is proposed to be x LexLIKE y [Threshold α] [InLanguages ...].

Suppose one is looking for books in different languages whose titles reference 'Manas Sarovar' (a holy lake in Tibet variously transliterated in English as 'Manas Sarobar', 'Manas Sarovar', 'Mansarovar', 'Manasarovar', 'Manasarowar', etc., written in Hindi usually as मानसरोवर, and in Bengali as মানস সরোবর). The following query

select BookTitle **from** BOOKS **where** BookTitle **LexLIKE** "Man% Sa[:syll:][vb]ar" Threshold 0.75
InLanguages Bengali, Hindi, English

should match the different renderings of 'Manas Sarovar' in the language sets indicated. Some examples of match using the LexLIKE operator for monolingual match in Bengali are পাখি LexLIKE পাখী, and পাখি LexLIKE পক্ষী Threshold 0.7.

The LexLIKE operator can be implemented if the grapheme to phoneme mappings (G2P) for the languages are specified. The G2P is used to convert the pair of expressions being compared to a canonical form. Approximate string matching can be performed in the phonemic space akin to the LexEQUAL operator [6].

In addition to crosslingual matching on exact words, crosslingual semantic matching is very useful for which the SemEQUAL operator had been proposed. We propose an equivalent notation for a crosslingual semantic matching operator which allows us to search a part of a field by using 'contains'. This is illustrated by the following example.

select BookTitle **from** Books **where contains**(BookTitle, [=rain=:syn])
InLanguages English, Bengali, German

will match with all BookTitles that contain synonyms of 'rain' from the documents in the specified languages. For example, this will match titles containing the word বৃষ্টি in Bengali, वर्षा in Hindi, and Regen in German.

4.5 Lexical processing

For lexical processing, we use the contains [4] predicate from Microsoft SQL server full text search. This allows us to match some of the words in a field. The contains predicate has the basic syntactic form **contains** (col list, '<search condition>'). This predicate is true if any of the indicated columns in the list col list contains terms which satisfy the given search condition. A search condition can be a regular expression denoting a word, a phrase, or can be a prefix, suffix, or stem. We propose the following lexical units that can be used:

`[:stem=book]`	matches all words with stem as 'book'.
`[:prefix=anti]`	matches all words with a prefix 'anti'.
`[:suffix=ment]`	matches all words with a suffix 'ment'.
`[:syn=book]`	matches all words synonymous to 'book' from the thesaurus.

Example:

select BookTitle **from** Books **where contains**(BookTitle, [:stem=discovery])

will match with all BookTitles that contain inflections of 'discovery', e.g., 'discoveries'.

select BookTitle **from** Books **where contains**(BookTitle, [:syn=discovery])

will match with all BookTitles that contain synonyms of 'discovery' in the thesaurus, like 'breakthrough' and 'uncovering'.

select BookTitle **from** Books **where contains**(BookTitle, [:stem=ভাষা] and [:suffix=কে])

will match with all BookTitles that contain a word which has stem ভাষা and has কে as one of its suffixes, and thus will match with a title containing the word ভাষাটাকেই, a word whose stem is ভাষা and which has three suffixes, টা, কে and ই.

5 Conclusion

In this paper we have examined different scripts and proposed different character classes to facilitate construction of regular expressions for different scripts. We also study the support that some databases provide for linguistic

matching with respect to the morphology of the words in free text, as well as semantic matching operators. Based on the study and the characteristics of different languages, we recommend that lexical classes for these units be used which can be the basis of matching. We have also discussed crosslingual matching operators that compare patterns across different languages. To check for semantic equivalence, semantic matching needs to be supported with the help of a thesaurus. Cross lingual semantic matching would require multi-language dictionaries and thesauri. However we have not discussed about the implementation of these operators in this paper. Many of these can be implemented on top of some existing database systems. For example, corresponding to every language, the appropriate character classes can be defined in the style file for the language. But the efficient implementation of some of the operators can be facilitated by intrinsic support by database engines, so that efficient index structures are created.

6 Acknowledgment

The author gratefully acknowledges the role of Professor Jayant Haritsa for the conception of this paper, as well as his help in all stages of preparing this paper.

References

[1] The Unicode Consortium. *The Unicode Standard Version 5.0*. Addison Wesley, 2006.

[2] Florian Coulmas. *The Writing Systems of the World*. Oxford, Blackwell, 1991.

[3] Peter T. Daniels and William Bright, editors. *The World's Writing Systems*. Oxford University Press, 1996).

[4] James R. Hamilton and Tapas K. Nayak. Microsoft sql server full-text search. *IEEE Data Engineering Bulletin*, 24(4):7–10, 2001.

[5] A Kumaran and Jayant R. Haritsa. Lexequal: Multilexical matching operator in sql. In *Proceedings of the 2004 ACM SIGMOD*, pages 949–950, Paris, France, 2004.

[6] A Kumaran and Jayant R. Haritsa. Lexequal: Supporting multiscript matching in database systems. In *9th International Conference on Extending Database Technology, EDBT 2004*, pages 292–309, Greece, 2004. Springer.

[7] A. Kumaran and Jayant R. Haritsa. Semequal: Multilingual semantic matching in relational systems. In *Database Systems for Advanced Applications, 10th International Conference, DASFAA*, pages 214–225, Beijing, China, 2005. Springer.

[8] Kevin Loney. Oracle database 10g: The complete reference, 2005.

[9] J.C. Wells. Orthographic diacritics and multilingual computing. In *Language Problems and Language Planning*, volume 24.3, Feb 2001.

[10] Wikipedia. Diacritic. http://en.wikipedia.org/wiki/Diacritic.

[11] Wikipedia. Writing system. http://en.wikipedia.org/wiki/Writing_system.

[12] Weiran Zhang. Pattern matching with multilingual regular expressions. In *24th Internationalization & Unicode Conference*, Atlanta, 2003.