LexEQUAL: Multilexical Matching Operator in SQL

A. Kumaran Jayant R. Haritsa*
Database Systems Lab, SERC/CSA
Indian Institute of Science, Bangalore, INDIA
{kumaran,haritsa}@dsl.serc.iisc.ernet.in

1. INTRODUCTION

To effectively support today's global economy, database systems need to store and manipulate text data in *multiple languages* simultaneously. While current database systems do support the management of multilingual data [4], they are not capable of *matching* data across languages with *different scripts* – for example, between English (Latin script), Russian (Cyrillic script) and Hindi (Devanagari script). As a first step towards addressing this lacuna, we recently proposed the LexEQUAL [5] operator for multilexical matching of *names*, which is of particular importance since a fifth of normal text corpora and a majority of search terms are proper and generic names [6].

Consider a hypothetical *Books.com* that sells books across the globe, with a product catalog as shown in Figure 1, where books in different languages are featured.

Author	Author_FN	Title	Price	Language
Descartes	René	Les Méditations Metaphysiques	€ 49.00	French
நேரு	ஜவஹர்லால்	ஆசிய ஜோதி	INR 250	Tamil
Σαρρη	Κατερινα	Παιχνίδια στο Πιάνο	€ 15.50	Greek
Nehru	Jawaharlal	Letters to My Daughter	$ 25.00	English
بهنسي ، د	عفيف	العمارة عبر التاريخ	SAR 75	Arabic
Nehru	Jawaharlal	Découverte de l'Inde	$ 9.95	French
寺井正博	著	秋の風 普及版	¥ 7500	Japanese
नेहरु	जवाहरलाल	भारत एक खोज	INR 175	Hindi

Figure 1: Multilingual *Books.com*

In this environment, the LexEQUAL operator can match an input name, such as "`Nehru`", across a user-specified set of languages with the SQL query syntax shown in Figure 2. The returned tuples (Figure 3), have in the Author column the multilexical strings that are *phonetically* close to `Nehru`. Note that the input name itself could have been given in any language.

*Supported in part by a Swarnajayanti Fellowship from the Dept. of Science & Technology, Govt. of India.

```
SELECT Author, Title, Language FROM Books
WHERE Author LexEQUAL 'Nehru' Threshold 0.3
IN { English, French, Tamil }
```

Figure 2: Sample LexEQUAL Query

Author	Title	Language
Nehru	Letters to My Daughter	English
நேரு	ஆசிய ஜோதி	Tamil
Nehru	Découverte de l'Inde	French

Figure 3: LexEQUAL Query Results

We present, in this demo, a prototype implementation of LexEQUAL on a standard relational database system, and demonstrate its viability, with regard to both result quality and computational efficiency, as a data integration tool for multilingual environments.

2. MULTILEXICAL MATCHING

The matching strategy of LexEQUAL is based on transforming the multilingual text strings to their equivalent *phonemic* representations, obtained using common linguistic resources, such as *text-to-phoneme* converters. The phoneme strings are processed in the canonical IPA alphabet [2] and in the Unicode [7] encoding format. Such phoneme strings represent a normalized form of proper names across languages, thus providing a means of comparison. Further, when the text data is stored in multiple scripts, this may be the *only* means of comparing them. Since the phoneme sets of two languages are seldom identical, we employ approximate matching techniques to match the phoneme strings. Thus, the multilexical comparisons are *inherently fuzzy*, making it only possible to produce a likely, but not perfect, set of answers with respect to the user's intentions.

To finetune the result quality to meet application requirements, the LexEQUAL operator is parametrized with regard to both the phonetic matching and the approximate matching. In [5] we demonstrated that for these tunable parameters, there exist appropriate ranges of choices that can be evaluated and customized for a given dataset, through which both good *Recall* and *Precision* may be simultaneously obtained.

The skeleton of the LexEQUAL matching algorithm [5] is given in Figure 4. Here, the transform function is imple-

```
LexEQUAL ($S_l$, $L_l$, $S_r$, $L_r$, $e$, $\mathcal{S_O}$)
Input:  Input Strings $S_l$, $S_r$
        Input String Languages $L_l$, $L_r$
        Match Threshold $e$
        Set of Languages for output $\mathcal{S_O}$
        Set of Languages with TTP Convertors $\mathcal{S_L}$ (global)
Output: TRUE, FALSE or NORESOURCE
1.  if $L_l \notin \mathcal{S_L}$ or $L_l \notin \mathcal{S_L}$ then return NORESOURCE;
2.  if $L_l \in \mathcal{S_O}$ then
3.      $T_l \leftarrow$transform($S_l$,$L_l$);
4.      $T_r \leftarrow$transform($S_r$,$L_r$);
5.      $Smaller \leftarrow (| T_l | \leq | T_r | \ ? \ |T_l| : |T_r|)$;
6.      if editdistance($T_l, T_r$) $\leq (e * Smaller)$ then
            return TRUE else return FALSE;
```

Figure 4: The LexEQUAL Algorithm

mented using standard linguistic resources, such as *Text-to-Phoneme* converters that convert a lexicographic string in a specific language to an equivalent phonemic string in the IPA alphabet. The editdistance function is a tunable function that can be made to compute the traditional *Levenshtein* edit distance or a modified distance metric, by parameterizing the cost functions for different edit operations. The *closeness* of the matching may be controlled by the *User Match Threshold* parameter, which specifies the level of allowable mismatch between the phonemic strings.

3. MIRA IMPLEMENTATION

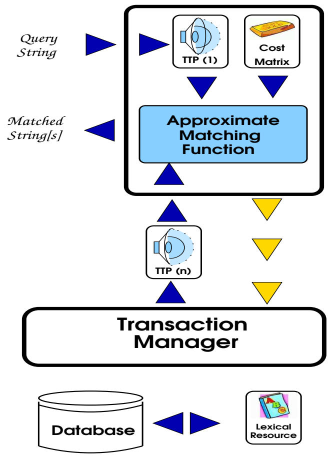

Figure 5: MIRA **Architecture**

Our multilingual query processing architecture – MIRA[1] for relational systems is as shown in Figure 5. For storage of multilexical strings and phoneme strings, we used a standard commercial database, namely *Oracle*, capable of storing and handling Unicode character set. The LexEQUAL and Edit-Distance functions were implemented as UDF's in PL/SQL language. The cost functions for computing the edit distance, were made as parameters to LexEQUAL function. For converting multilingual strings in specific languages to their equivalent phoneme strings, *Text-to-Phoneme* converters are used when available, or by referring to standard linguistic resources, such as *Oxford English Dictionary*, otherwise.

[1] Multilingual Information processing on Relational Architecture

While the standard implementation of LexEQUAL as a UDF was slow, we were able to improve the efficiency of matching by utilizing either Q-Gram filters [1] or Phoneme Indexing techniques [3, 8], that weed out most of the *false positives*, thus optimizing calls to the more expensive UDF function. Further performance improvements could be obtained by internalizing our "outside-the-server" implementation into the database engine.

3.1 Salient Features of MIRA Architecture

Modular Design The language resources, such as *text-to-phoneme* convertors, may be added easily, as the functions that are called based on a table lookup. Our design goal is to make the database capable of processing a new language, just by installing a few resources and adding references in the look-up table.

Approximate Matching Customization The *closeness* of match may be controlled using the Threshold parameter, depending on the requirements of the application. A *Homeland Security* application may need tighter matches than a *Directory Enquiry* application.

Phonetic Matching Customization The matchable set of phonemes may be customized, by forming clusters of *near-equal* phonemes. Such clusters may be installed based on inputs from linguistic community.

4. DEMONSTRATION

The multilexical operator LexEQUAL is a first step towards a transparent support for multilingual functionality in database systems. In this demonstration, we present an outside-the-server implementation on the *Oracle 9i* (Version 9.1.0) database system, running on a standard Pentium/Windows NT platform. The LexEQUAL and the approximate matching functions are implemented as UDF's in PL/SQL programming language. The user interface is developed in Java environment. We demonstrate LexEQUAL's effectiveness with a sample data set containing generic and proper names from English and Indic domains.

5. REFERENCES

[1] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan and D. Srivastava. Approximate String Joins in a Database (almost) for Free. *Proc. of the 27th VLDB Conf.*, 2001.

[2] The International Phonetic Association. *http://www.arts.gla.ac.uk/IPA/ipa.html.*

[3] D. Knuth. The Art of Computer Programming, Volume 3: Sorting and Searching. *Addison-Wesley*, 1973.

[4] A. Kumaran and J. R. Haritsa. On the Costs of Multilingualism in Database Systems. *Proc. of the 29th VLDB Conf.*, 2003.

[5] A. Kumaran and J. R. Haritsa. Supporting Multiscript Matching in Database Systems. *Proc. of the 9th EDBT Conf.*, 2004.

[6] M. Liberman and K. Church. Text Analysis and Word Pronunciation in TTS Synthesis. *Advances in Speech Processing*, 1992.

[7] The Unicode Consortium. *http://www.unicode.org.*

[8] J. Zobel and P. Dart. Phonetic String Matching: Lessons from Information Retrieval. *Proc. of 19th SIGIR Conf.*, 1996.