On Database Support for Multilingual Environments

A. Kumaran* **Jayant R. Haritsa**

Database Systems Laboratory
Indian Institute of Science
Bangalore 560012, INDIA

Abstract

Global e-Commerce *and mass-outreach* e-Governance *programs have brought into sharp focus the need for database systems to store and manipulate text data efficiently in a suite of natural languages. While some means of storing and querying multilingual data are provided by all current database systems, to the best of our knowledge there has been no prior study of their functionality or efficiency in this regard. In this paper, we explore the multilingual support needed by the user community and what is currently provided by the popular database systems to satisfy these needs. Specifically, a comparison of multilingual features supported by the database systems is provided against a set of relevant parameters. Initial results from our performance study indicate that serious lacunae exist in the performance with respect to multilingual data. We propose a new data type and associated database system architecture components for making the performance of the database system to be language independent. Results from our initial implementation of the proposed methodology are encouraging indicating the value of such an approach.*

1. Introduction

The rapidly accelerating trend of globalization of businesses and the success of *e-Governance* solutions require data to be stored and manipulated in many different natural languages. As the primary data repository for such applications, database systems need to be efficient with respect to multilingual data. While all current commercial and open-source database systems support some means of storing and manipulating such data, to the best of our knowledge there has been no prior study of their functionality or efficiency in this regard. This paper explores the multilingual support needed by the user community and the features provided by popular database systems to satisfy the same. We define a set of parameters in the multilingual arena and compare how the popular database systems measure up with respect to these parameters. We also provide some initial results from our performance study, which indicate that serious lacunae exist in performance with respect to handling of multilingual data. We propose a new data type and enhancements to the database architecture to handle multilingual character sets efficiently and equitably.

The remainder of this paper is organized as follows: Section 2 defines a set of requirements to be supported by the databases with appropriate examples. Section 3 provides a survey of database systems support for the above requirements and provides some preliminary results from our performance experiments. Section 4 enumerates possible research avenues for the database community to provide efficient multilingual support for the users.

2. User Requirements for Multilingual Support in Database Systems

In this section we specify the requirements of users of multilingual databases, with examples from typical applications.

2.1 Storage and Querying Requirement

Among the primary drivers for the need of multilingual information is the phenomenal growth of the Internet and its impact on global *e-Commerce* and *e-Governance* solutions for mass outreach. The volume and usage of such systems critically require the multilingual data to be stored and manipulated efficiently.

Consider *Bhoomi* [3], one such real-life e-Governance system of the State of Karnataka in India. *Bhoomi* is a computerized land records system storing about 20 million land records of rural farmlands in the state. The data is stored in the local language of the state, *Kannada*, as the system

*Contact Author: kumaran@dsl.serc.iisc.ernet.in

0-7803-7868-7/03/$17.00 © 2003 IEEE.

is intended to provide friendly access to the farmers of the state. Efforts are underway in different states to develop information systems along the lines of *Bhoomi*, in the respective regional languages. Records from a hypothetical national database that integrates information from all such regional databases may resemble those in Figure 1.

Last Name (Multilingual)	First Name (Multilingual)	Survey ID (English)	Address (Multilingual)	PAN# (English)
[illegible]	[illegible]	DK 99/76 PS 199/81-82	475, HAL [illegible], [illegible]560008	AAZ9847G
Roy	Arundhati	UBDK 90/54 UY 121/66-67	1, Blue Lotus Lane, Goa 403001	AAY97182
[illegible]	R. K.	TNJ 1872/ OLD 20-21	7, [illegible] 613001	AAB3489W
नेहरू	जवाहरलाल	LK 62/DNSK 112.12M	१०, मोदि मार्ग, दिल्ली ११०००१	NULL
Müller	Pasçalè	JUH 1899/1ANB 1927.28	8, Orange County, Coorg 571234	BBC7619X

Figure 1. Sample Records from a National Land Records Database

The basic multilingual requirement is that the database system must be capable of storing data in different languages. While in specific instances it may be necessary to restrict the data stored in a column to a single language type, it may not always be possible or desirable to make such restriction universal. In the example above, text strings in different languages may be stored in the same column and a multilingual string may contain characters from different languages.

The data must be queryable using query strings in any of the languages and SQL language primitives must support such requirements. The need for having query interface itself in different languages is not specified as a requirement and is left for individual user communities to design and implement. The output of the query could be multilingual and in such cases the presentation order must be intuitive and as per conventions specified in those languages. From database point of view, proper sorting of multilingual strings as per local conventions is a necessity both for proper user output and for internal database processing, such as index building. The user interface issues are not specified, as the database handles text strings in their *logical order* [5] only. Formally, the *Storage and Query requirement* may be stated as:

The storage and queryability of multilingual data must be as intuitive as those in default database character set; the output must be presented as per the conventions of the multilingual script.

2.2 Interoperability Requirement

The multilingual data stored in a database must be meaningful for other systems as well. For example, the records of the Land Records database shown in Figure 1 must be available to other systems in a format that is recognizable by those systems. Though proprietary formats may be specified and fine tuned for the requirements of specific applications usually the interoperability suffers, and hence such proprietary formats must not exist in an increasingly multilingual world, at least not at the interface level. Formally, the *Interoperability requirement* may be stated as:

The multilingual data must be stored in such a format that it is interchangeable with other information systems transparently.

2.3 Language Independence Requirement

We expect that global e-businesses such as *Amazon.com* would be providing customized service to their customers in the regional languages in due course. Given that under such customization, the pages need to be generated with multilingual data dynamically at the access time, the systems must be equally efficient in any of the languages of choice. The prime requirement here is that a user should not be hampered by the language of his or her choice; that is, the performance of the database for two languages must be identical, if the size of the repertoires are the similar. Though efficiency is a well accepted fact, we state it explicitly as follows:

Access and processing of the multilingual data must be efficient and independent of the type of language stored and processed.

2.4 Lexical Processing Requirement

While [in]equality of textual information is well understood within a single script, we strongly believe that equivalence across languages also must be supported. Consider the following requirement of *Government of India*: A citizen of India is required to file a Tax Return only if he has both a land registration and a telephone subscription in his name (*This simple case is culled out of a real and more complex requirement*). Such people who satisfy both requirements can be enumerated by joining the records from the Land Records database shown in Figure 1 with records from the Telephone Subscriber database, which is usually in English, as shown in Figure 2.

The query to get the potential tax-payers needs to join multilingual name attributes from the Land Records database with English name attributes from Telephone Subscriber database (*and join perhaps other salient demographic attributes not shown here*), as shown below:

```
Select T.FirstName,T.LastName,T.Address
From Land L, Telephone T
Where L.FirstName = T.FirstName
  and L.LastName = T.LastName;
```

Tel #	Last Name	First Name	Address	Type
5253367	Bharati	Narayan	475, HAL Layout, Bangalore 560038	RES
3942737	Tata	J.R.D.	1, Sir CV Raman Road, Bangalore 560012	DSL
5295510	Turing	Alan	1912, Cambridge Ave, Bangalore 560008	DSL

Figure 2. Sample Records from Telephone Subscriber Database

Such need to integrate data from diverse character sets is amplified further when one considers international organizations such as Interpol or UNESCO, which handle data in any/all of the world's languages. We refer to such cross-script joins as *Lexical Joins*. Clearly, such comparison requires a notion of equivalence between characters from different scripts. We specify such a *Lexical Join requirement* as follows:

Character strings in different scripts may need to be compared using pre-defined lexical mappings between the characters of those scripts.

2.5 Linguistic Processing Requirement

Joining on attributes containing data from different languages need not be restricted to lexical level only, but may be extended to meaning of individual data items as well. Suppose, in the above example, identification of potential tax payers require comparison of an additional demographic attribute, *Gender*. The values for such attribute may be specified differently in different languages (and hence neither *equal* nor *equivalent lexically*), but they are all equivalent linguistically to one of $\{Male, Female\}$. In such cases, matching of data requires a linguistically enhanced join operator, which may match data items across languages using linguistic resources such as Dictionaries or Thesauri.

We refer to such cross-language joins on meanings of attributes as *Linguistic Joins*. The requirement for *Linguistic Join* may be formally stated as:

Data values from different languages may need to be compared using pre-defined linguistic mapping between words or phrases of different languages.

However, we would like to emphasize here that linguistic processing is a fertile discipline on its own. We propose the integration of such linguistic technologies with databases to serve the needs of the users. The specification of exact requirements for such integration is open-ended and is beyond the scope of this paper. However, we recognize that such integration of Linguistic and Database technologies will happen in due course and the simple *Linguistic Join* operator outlined here may be a first step in that direction.

3 Current Support for Multilingual Data in Databases

We start this section with some background information that may be needed to understand the multilingual issues. Next, a brief outline of the support specified in the SQL standards for processing of multilingual data is provided. For comparing popular database systems, we chose a set of parameters that are relevant and highlight the support provided by each database system for this suite of parameters. Subsequently, we provide a summary of how the requirements outlined in Section 2 are satisfied by the database systems considered. We conclude the section with some sample results from our multilingual performance experiments.

3.1 Background Concepts

In this sub-section, we provide some basic concepts in encoding lexical data. An informed reader may skip this section and go directly to Section 3.2.

3.1.1 Character Set and Encoding

A *Character* is thought of as the smallest component of written language that has a semantic value. The set of all the characters in a language is called a *Repertoire*. A *Character Encoding* assigns a unique value to each of the characters in a repertoire. There are several well-known encoding, such as ASCII, ISCII [1], ISO-8859 [7] and Unicode [5], that form the basis for storage and interchange of text data among computer systems. While ISO-8859 based character sets are the most widely used currently, Unicode is becoming a defacto standard for global interchange of information.

3.1.2 Unicode Encoding

Unicode [5] is a universal character encoding standard that allows storage of characters from any known alphabet or ideographic system, derived from the ISO10646 standard [8], called *Universal Character Set* or $UCS-2$. UCS-2 provides a unique 2-byte code for every character, no matter what the platform, programming environment or language. Unicode has allocated encoding for every character along the same lines as UCS-2. The encoding are arranged in *Character Blocks*, which encodes contiguously the characters of a given repertoire, typically characters in a single script. The characters from a code block may support multiple languages, but usually a single language may be served by a single code block only. Unicode also specifies 3 different byte encoding ($UTF-8, UTF-16$ and $UTF-32$) to store the same character codes, but in a byte, word or double word oriented formats. Each of these encoding are equivalent and can be transformed in to each other by simple, fast

bit-wise operations. A vendor is free to choose any of the above three encodings to be fully compliant with Unicode.

Lang.	Encoding	String	Representation (Hexadecimal)
English	ASCII	Narayan	E4.16.27.16.97.16.E6
English	Unicode (UTF-16)	Narayan	00.E4.00.16.00.27.00.16.00.97.00.16.00.E6
English	Unicode (UTF-8)	Narayan	E4.16.27.16.97.16.E6
Kannada	ISCII	ನಾರಾಯಣ್	A8.BE.B0.BE.AF.A3.CD
Kannada	Unicode (UTF-16)	ನಾರಾಯಣ್	0B.A8.0B.BE.0B.B0.0B.BE.0B.AF.0B.A3.0B.CD
Kannada	Unicode (UTF-8)	ನಾರಾಯಣ್	E0.AE.A8.E0.AE.BE.E0.AE.B0.E0.AE.BE.E0.AE.AF.E0.AE.A3.E0.AF.8D

Figure 3. Sample Encoding in Various Formats

Figure 3 illustrates character representation of equivalent multilexical strings in ASCII and Unicode encodings. It should be noted that the UTF-8 encoding preserves ASCII encoding, while tripling the size of Indic strings from their proprietary ISCII encoding. The UTF-16 encoding doubles the size of data for both ASCII and ISCII strings.

3.2 What does the SQL Standard offer?

Until the SQL-92 [12] standard, there was not much support specified in relational databases for languages other than English, which was assumed as a default. However, in late eighties the need for supporting multiple character sets was recognized and specifications were introduced in the standard to overcome this deficiency.

In the multilingual arena, the SQL-92 Standard supports the specification of a data type to store multilingual characters, called NATIONAL CHAR (also referred to as *NChar*) that is very similar to character data type but wide enough to hold multilingual data. A table column may be specified as an NChar type and characters from any national character set may be stored in such a column. Also, since the national character set may sort differently from default database character set, the SQL standard allows the specification of *collation sequences* to correctly sort and index the data. Significantly, the format of storage of national character set is left unspecified, and the database vendors are free to choose any format for storage. Specifications are also provided for restricting a NChar column to store characters only from a specified repertoire. The standard specifies that comparison of two NChar strings is valid only with respect to a repertoire and considers comparison across repertoires as binary comparison, with the assumption that comparison of characters across repertoires is meaningless.

Finally, even the recently released SQL standard – called SQL:1999 [13], has not gone beyond SQL-92 in the area of multilingualism.

3.3 What do Popular Databases offer?

In the academic and research community, a few proprietary multilingual database systems have been developed and deployed, such as [9] and [11]. While these systems are extensive in their lexical and linguistic capabilities, their applicability is limited to specific domains. Therefore, in this paper, we focus primarily on the popular general purpose database systems, such as Oracle 9i (9.0.1), Microsoft SQL Server 2000 (8.00.194), IBM DB2 Universal Server (7.1.0) and MySQL (4.0.3-*Beta*).

In the following sub-section, we specify a variety of parameters to evaluate multilingual support and assess how these databases measure up on these parameters. Only the parameters that directly impact database processing are selected for comparison. We would like to emphasize that issues such as *Internationalization/Localization* that refer to the process of making a piece of software portable and customisable across languages and *Layout/Rendering* that deal with display of multilingual text for the user interfaces are *not* considered, as these do not impact database processing. However, they share some common resources with databases, such as *Locale*.

3.3.1 Storage Format of Multilingual Data

While the 8-bit ISO-8859 based character sets are the default character sets in most database systems, the main issue with them is that their width is not sufficient to store multilingual data. However, most database systems have taken either Unicode or UCS-2 as the storage format for implementing NChar data type. While Oracle 9i and DB2 have allowed user specification of NChar as one of UTF-8 or UTF-16, SQL Server stores NChar as UCS-2. The open-source MySQL plans to add support for Unicode, though this feature is not available as yet.

While Unicode achieves a much-needed standardization for interoperability, there may be undesirable side effects resulting from improper user choice of the storage format for NChar. Those databases that allow UTF-8 format may offer a better space efficiency for data that is dominated by ASCII-based scripts, whereas the same UTF-8 format may triple the size of the database for data that is predominantly in Indic scripts. The UTF-16 encoding doubles the size of the database in both the cases. The increased space directly translates to increased system cost and also has adverse impact on the query performance. However, the storage size also depends on whether the database system uses the specified format for the storage or has implemented some internal optimizations.

3.3.2 Collation Sequences

The *Collation sequence* is fundamental to most database operations, such as comparison, sorting and indexing. Unicode consortium has specified the semantics of comparing two Unicode strings in [6]. Briefly, this collation algorithm makes use of three levels of sorting, based on the base characters, base character plus the diacritical marks or the combination of the base characters, diacritical marks and the case of the letter. The collation algorithm also provides support for additional comparison levels that can be specified by users. If no sort sequence is specified for a multilingual column, the sort order is taken to be binary.

All the commercial databases support Unicode collations along with all three levels of comparison. Oracle has about 50 pre-defined collations while DB2 has about 40 pre-defined collations. However, users must use only one of these pre-defined collations. SQL Server uses collations defined in the underlying Windows OS, thus providing a tighter integration with other language handling components of the system. MySQL has pre-defined about 23 collations and also allows users to define new collations through source-code changes. While flexible, this approach requires source knowledge and expertise and may lead to potential inconsistencies. Oracle and DB2 also support multilingual sorts, which allow sorts of a mixed language strings from a limited set of languages. Though user-specified collations are allowed in SQL standards, no commercial database systems has implemented this feature.

3.3.3 Multilingual Data Indexing

Collation sequences are used to build indexes on specific attributes. All the databases support indices on multilingual data using one of the pre-defined collation sequences. Oracle and DB2 allow multiple indices on the same column using different collations allowing the same data to be processed with different language conventions. It is not clear from our reading whether SQL Server supports multiple indices.

3.3.4 Lexical / Linguistic Query processing

When we consider query processing with language data the differences between *Database Systems* that focus on representation and efficient manipulation and *Natural Language Processing* that focuses on semantic content, are brought into focus. However, these disciplines are complementary to each other and may symbiotically provide enhanced service to the users in Internet era.

Query processing in multilingual environments could vary from being a simple string matching (in different scripts) to a complex semantic query, by considering orthogonal variations of transliteration or translation of query and stored data, semantic or thematic querying, and cross-language retrieval using richer linguistic resources such as Wordnet [2].

All the lexical and linguistic query processing require varying amounts of linguistic processing; since no linguistic processing is specified in SQL standards, each vendor has taken their own approach for handling such queries, making comparison between them difficult. MySQL has a very rudimentary support for natural language queries, but plans to add linguistic processing to the server. SQL Server provides linguistic analysis and querying in a handful of languages. DB2 has integrated with normal SQL, text processing features that offer a rich set of linguistic features for query processing. Features include linguistic indexing of data using morphological and other linguistic analysis tools and retrieval using semantic matching of query keywords. Oracle's Text Server Option provides a similar set of features, enhanced by rich indexing schemes. However, these advanced capabilities are limited to documents in only a handful of languages – primarily Western European and a few East Asian languages. However, each vendor has plans to add more languages in the future versions.

3.3.5 Summary of Multilingual Support by Commercial Systems

The comparison of features discussed in the preceding sections is summarized in Table 1. Keeping in mind those requirements that are specified in Section 2, we observe that in general all the database systems have implemented equivalent support for multilingual *Storage and Querying* requirement using a wide NChar format and NChar predicates that are equivalent to Char predicates. The commercial database systems support Unicode or UCS-2 for *Interoperability* requirement, while MySQL has promised support for Unicode soon. The question of how efficient the database systems are in supporting multilingualism - the *Language Independence* requirement, is explored in the Section 3.4.

The support for *Lexical Processing* is not available in any of the database systems yet, as all have assumed that comparison across scripts is meaningless. We explore this requirement in our research agenda in Section 4. Support for the *Linguistic processing* requirement is not uniform among the databases, due to the fact that SQL Standards have not specified guidelines on these features yet. However, a rich set of features are provided by all commercial databases for linguistic querying of underlying data, though such capabilities are currently restricted to a handful of languages.

3.4 Multilingual Performance Analysis

To quantify the performance of the database systems with respect to handling of multilingual text data, we conducted a set of experiments on a popular database system

Database	Oracle 9i Internet Server	Microsoft SQL Server2000	IBM Universal Server	MySQL
Storage Format	Supports Unicode 3.01 NChar as UTF-8 or 16	Supports Unicode 3.01 NChar as UCS-2 Supports UTF-8 for XML	Supports Unicode 3.01 NChar as UTF-8 or 16	No Unicode support yet.
Collation Sequence	Complies to Unicode Standards All 3 Unicode levels with pre-specified; Supports cross-lingual sorts	Complies to Unicode Standards All 3 Unicode levels pre-specified; Integrated with collations from OS	Complies to Unicode Standards All 3 Unicode levels pre-specified; Supports cross-lingual sorts	Collations are user definable. by *source changes.*
Indexing	Index using Collations Multiple Indexes	Index using Collations	Index using Collations Linguistic Indexes	Index using Collation
Query Processing	NChar can use all Char predicates *Implicit conversions used in predicates with* CHAR & NChar	NChar can use all Char predicates	NChar can use all Char predicates *Implicit conversions used in predicates with* CHAR & NChar	Supports rudimentary *Natural Language Query facility*
Locale	About 50 Locales pre-specified	Uses all Locale specified in Windows OS	About 40 Locale pre-specified	About 23 Locale pre-specified

Table 1. Comparison of Commercial Databases vis-a-vis Multilingual Support

with two different data sets; the first data set contained data in ASCII and the second contained equivalent Unicode data in Indic scripts in the popular UTF-8 encoding. Data sets of about 240 MB size were generated using a modified TPC-H data generator and loaded onto the database system under study. The tests were run on a standard Pentium 1.7GHz machine with 512MB memory. Carefully chosen queries that approximate the performance of standard relational operators were run. Typical experiment involved measuring running time for equivalent queries involving integers (for establishing a baseline), Char and NChar text. A sample of run times from our initial experiments with one of the database systems is provided in Table 2. Space-wise, we observed that the storage needed for NChar data is nearly twice that of equivalent Char data.

Relational Operator	Integer Data *(Sec)*	Char Data *(Sec)*	NChar Data *(Sec)*	Operator Slowdown *(Char vs NChar)*
Table Scan	8	9	26	188%
Index Scan	0.11	0.12	0.33	165%
Join	27	97	171	76%

Table 2. Performance of Relational Operators

We observe that under default parameters for the machine, OS and the database, the multilingual queries are significantly slower, as shown in Table 2. Clearly, such inefficiencies in the basic relational operators are bound to affect overall query performance. Further, what is more worrisome is the fact that we observe that the optimizer *is not correctly* estimating such slowdown, which could potentially have a major impact on query performance by allowing inefficient plans to be selected.

4 A Research Proposal for Multilingual Support in Databases

So far in the paper, we have highlighted the requirements from the user community and the support provided by the popular database systems, vis-a-vis multilingual data. All gaps between the two must be addressed by the database research community and in the remainder of this paper we discuss three important research issues that need to be addressed for wider adoption of multilingual databases: *lexical and linguistic feature enhancements* in databases, *benchmark suites* for feature and performance analysis, and *database architecture components* for efficient support for multilingual data.

4.1 Lexical and Linguistic Features

4.1.1 *Lexical Join* Operator

As per SQL-92 standard, comparison of two strings is considered to be meaningful only if they are from the same repertoire. Since NChar does not contain the repertoire information the comparison of two NChar strings is primarily considered as a binary comparison. Clearly, this restriction

has an impact on *Lexical Processing* Requirement given in Section 2.

Equality comparison of strings from different languages makes sense for proper nouns, though we recognize that such comparisons may be limited to strings from languages within an equivalent set of languages. While the definition of the equivalence sets of languages and equivalence of individual characters in a given pair of languages are left to linguists, we maintain that such equivalence once defined, may be used for lexical joining of data.

We believe that there is value to such lexical comparisons and suggest that SQL extensions may be defined for such comparisons; further, we recommend that it be included in the future SQL standards.

4.1.2 *Lingual Join* Operator

The lexical matching capabilities of database systems using *Lexical Join* may be extended further to matching on meaning of attributes as well. We propose another new join operator, tentatively called *Lingual Join*, to match on semantic values of attributes using generic, multi-purpose linguistic resources, such as WordNet [2]. The necessary linguistic resources that map equivalent concepts between pairs of languages must be defined by linguists and be taken as input for implementing *Lingual Join* operators.

Given that the linguistic resources such as WordNet need to be modeled as dense graphs, storing them in relational database systems parallels the well-known efforts in the area of mapping of data between XML and relational formats as illustrated in [15]. Further, availability of such rich linguistic resources in multiple languages in the database systems may be useful for linguistic researchers as well.

4.2 Performance Benchmarks

Though traditionally the databases are used for large amounts of enterprise data, multilingual text is becoming a major component of the database storage today. While several benchmarks such as TPC benchmarks [4], are available for comparing performances of databases with respect to traditional data, none exists for measuring efficiency of databases with respect to multilingual data, to our knowledge. It is our belief that such performance differentials as highlighted in Table 2 will exist in most database systems, though the extent of such deviations is unknown at this time.

All such observations point to the need for a well-accepted and well-trusted framework for comparing different database systems, to aid the users in selecting an appropriate database system for their needs. Such a benchmark should test overall functionality and performance of the database systems and performance of crucial system components such as Query Optimizer.

4.3 A Proposed Data type – *LChar*

Our initial analysis of performance results suggests that the differences in performance are primarily attributable to the increased storage needed for multilingual data. While Unicode provides interoperability, it has an adverse effect on storage. Hence, it is essential to find a way of reducing the storage space needed without compromising Unicode standards.

We outline here our approach to reduce the space overheads for Unicode strings that is consistent with Unicode standards. We propose a new data type – *LChar*, which stores a given Unicode string as two pieces internally; the first piece storing the code block of the string as the meta data for the the second piece that stores the offsets for every character in to the code block. This approach stems from our observation that while most Unicode code blocks contain less than 256 characters (thus requiring only one byte for storage of the offset), the default 2-byte representation is used for storing each character in UTF-16. Given that a data item is most likely to be in a single language, the bits encoding the code block are merely repeated for each and every character that is a part of the text string. The corresponding Unicode string may be generated on demand at memory speeds, by combining the meta-data (*code block information*) with the data string (*offsets*), using a simple and efficient bit-wise operation.

4.4 A Proposed Database Architecture for Multilingual Environments

Assembling all the pieces above, we propose a set of database architecture components for efficient processing of multilingual data, as shown in Figure 4. Our proposals are highlighted by shaded boxes in the figure.

We propose that the new data type defined above – *LChar*, be implemented as the storage format for multilingual characters. Such an implementation would be efficient storage-wise and would also satisfy the *Language Independence* requirement. To support LChar data type, the following changes to the database architecture are needed: Database catalog must be enhanced to model LChar data type and proper schemes must be devised to efficiently store and process the split representation of LChar strings. The query processing module must implement changes to Parser to take into account the enhanced SQL syntax and for converting input Unicode strings to LChar strings. The Optimizer and Code Generator must be modified to take into account the mapping of the user query to an internal query that handles the split image of LChar strings for a given Unicode string. Changes must be made in optimizer modules to model the costs associated with new LChar data type accurately, to aid the proper query plan selection. Further,

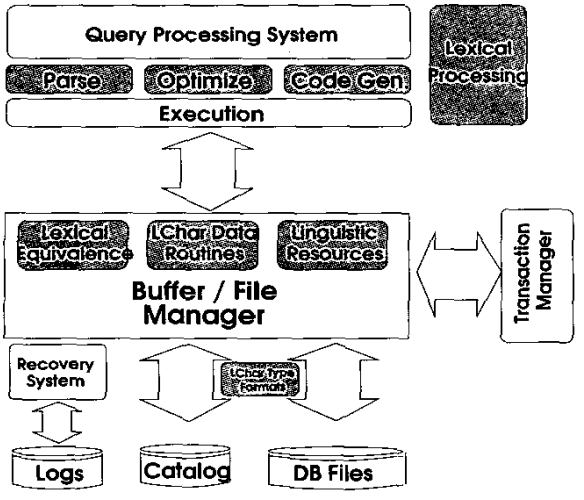

Figure 4. Architecture

optimizer mis-estimate of queries with NChar data type is a major weak point that we found in our initial experiments. Buffer and File management modules in the core of the database server must be enhanced with the new LChar data type, by implementing efficient bit-wise operations to convert strings between Unicode and LChar. Semantics of conversions between LChar and other database data types must be defined, though we expect them to be very similar to those of Unicode based data type.

Most importantly, the database engine must be modified to store the lexical resources to implement *Lexical Join*. The mapping tables between pairs of languages must be stored in main memory for efficient access, as we expect the mapping tables to have a small footprint.

We propose wider adoption of linguistic technologies and implementation of *Linguistic Join*, using linguistic resources. Resources such as WordNet [14] may be useful in comparing meanings of words in different languages, if a proper synset mapping is available between WordNets of different languages. The availability of such resources in different languages will help to make implementation of linguistic operators possible.

5 Conclusion

In this paper we presented a set of requirements from the user community for multilingual database systems and justified the same with examples from typical *e-Commerce* and *e-Governance* solutions. We provided a survey of the support offered by popular database systems to satisfy such requirements. We find that the database systems have taken a near uniform approach in supporting storage and querying requirements by supporting Unicode or UCS-2. However, wide gaps exist in the performance aspect, as suggested by our preliminary experiments with a popular database. Serious space overheads and differences in the performance of standard database operators working on equivalent data sets in Char and NChar underscore the need for a comprehensive performance study and performance improvements. Further, we see that some of the requirements of user community to merge data lexically and linguistically from different languages is not satisfiable by current SQL standards.

We propose a comprehensive solution to satisfy these needs by adding a new data type as well as new processing components to the basic database architecture. We suggest that the new operators outlined here be considered for inclusion in the future versions of SQL standards as a uniform mechanism to combine multilingual data. We are currently engaged in a comprehensive study of all the issues raised in this paper and full details of our results will be made available in [10].

References

[1] *http://tdil.mit.gov.in.*

[2] *http://www.cogsci.princeton.edu/wn.*

[3] *http://www.revdept-01.kar.nic.in/Bhoomi/Home.html.*

[4] *http://www.tpc.org.*

[5] *http://www.unicode.org.*

[6] M. Davis. Unicode collation algorithm. *Unicode Consortium Technical Report*, 2001.

[7] ISO. ISO/IEC 8859 Information Processing – 8-bit Single-Byte Graphic Coded Character Sets. *ISO/IEC 8859-15:1999, 1999.*

[8] ISO. ISO/IEC 10646-1:2000, Information Technology – Universal Multiple-octet Coded Character Set (UCS) – part 1: Architecture and Basic Multilingual Plane. *ISO/IEC 10646-1:2000*, 2000.

[9] R. King and A. Morfeq. Bayan: An Arabic Text Database Management System. *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, 1990.

[10] A. Kumaran and J. R. Haritsa. Bridging the Digital Divide Between Database and Linguistic Technologies. *IISc/Database Systems Lab Technical Report* (forthcoming), 2003.

[11] C. Lu and K. Lee. A Multilingual Database Management System for Ideographic Languages. *Chinese University of Hong Kong Technical Report*, 1992.

[12] J. Melton and A. R. Simon. *Understanding the New SQL: A Complete Guide.* Morgan Kaufmann, San Francisco, California, 1993.

[13] J. Melton and A. R. Simon. *SQL 1999: Understanding Relational Language Components.* Morgan Kaufmann, San Francisco, California, 2001.

[14] G. A. Miller. Wordnet: A Lexical Database. *Communications of the ACM*, 38:11:39–41, 1995.

[15] J. Shanmugasundaram *et al.* Relational Databases for Querying XML Documents: Limitations and Opportunities. *Proceedings of the 25th VLDB Conference*, 1999.