MULTILINGUAL DATA MANAGEMENT IN DATABASE ENVIRONMENT

***Abu Sayed Md. Latiful Hoque*[1] *and Mohammad Shamsul Arefin*[2]**

[1]Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology (BUET), Dhaka-1000, Bangladesh.
Email: asmlatifulhoque@cse.buet.ac.bd
[2]Department of Computer Science and Engineering,
Chittagong University of Engineering and Technology (CUET), Chittagong-4349, Bangladesh.
Email: sarefin_406@yahoo.com

ABSTRACT

Global E-Commerce and E-Governance programs have brought into sharp focus for the need of database systems to store and manipulate data efficiently in a suite of multiple languages. While existing database systems provide some means of storing and querying multilingual data, they suffer from redundancy proportional to the number of language support. In this paper, we propose a system for multilingual data management in distributed environment that stores data in information theoretic way in encoded form with minimum redundancy. Query operation can be performed from the encoded data only and the result is obtained by decompressing it using the corresponding language dictionaries for text data or without dictionary for other data. The system has been evaluated by both syntactic data and real data obtained from a real life schema. We have compared the performance of our system with existing systems. Our system outperformed the existing systems in terms of both space and time.

Keywords: *Multilingual Data Management, database environment, data dictionary, lexeme, query*

1.0 INTRODUCTION

Efficient storage and query processing of data spanning multiple natural languages are of crucial importance in today's globalized world [1, 2, 3]. As Internet has become a primary medium for information access and commerce, multilingual data management [4, 5] in database environment can be treated as a vital issue for the availability of information in the native language of the Internet users. Survey results indicate that the demographics of the Internet are steadily becoming multilingual. The non-native English speaking users of the Internet has grown from about half in mid-90's, to about two-thirds now and it is assumed that the majority of the Internet information will be multilingual by 2010 [6]. It has been found that a user is likely to stay twice as long at a site and four-times more likely to buy a product or consume a service, if the information is presented in their native language [7]. Hence, it is important that the information systems support efficient handling of multilingual data.

Research results [8] show that significant performance degradation occurs when handling multilingual data using current database systems. An efficient Multilingual Data Management System (MDMS) is necessary to overcome the limitations of multilingual data handling capability of the existing database systems and for better searching and browsing capabilities in different languages, accessing information stored in different languages, accelerating globalization of businesses and implementing e-Commerce and e-Governance.

We have to consider generally three main considerations for MDMS. Firstly, there should be a technique by which the data will be represented in a language-independent way. Secondly, an efficient translator is needed for performing translation among languages. Thirdly, an efficient mechanism is required to perform different types of multilingual operations in a distributed database environment [9]. These are the crucial issues in multilingual data management.

This paper presents a system to store multilingual data in a language independent way such that database evolution is easier. Queries can be performed using a translator-based approach. In this approach when information in a specific language is provided, the translator will generate its corresponding information in the target language. Schema evolution which is difficult in the existing systems, is simple and easier in this system to maintain database consistency. Query performance is also significantly faster.

2.0 LITERATURE REVIEW

While a rich body of literature on multilingual information processing exists in the Natural Language Processing [10] and Information Retrieval [11, 12] communities, there is comparatively very little in the database context. In database literature, the multilingual data management issues may be classified as solutions for specific languages, data integration solutions or proprietary solutions.

A database system for handling Arabic data is presented in [13]. This work presents specific issues and solutions for storing, indexing, querying and presenting Arabic language data, in an object oriented way. A database system has been presented for storing and query processing ideographic Chinese, Japanese and Korean (CJK) data in [14], where the authors primarily focus on the definition of resources needed for handling ideographic scripts in database systems. Though both these papers address issues specific to the languages concerned (Arabic and CJK languages respectively), neither of them propose solutions for multilingual data management for interchange of information among users of different languages.

The Federated Multilingual Database system (FEMUS) [15] can integrate data from different data models and the associated query languages. But the limitation of this system is that it does not address issues in integrating data from different natural languages. A multilingual query-processing framework for sharing lexical resources is discussed in [16]. The main focus of this work is on improving the efficiency of administration of multilingual resources in database environment. But it does not provide any guideline for multilingual query processing.

The Look-Alike-Sound-Alike [17] and EROS [18] are two application specific multilingual data management systems. These systems support multilingual data for specific domain but do not address general-purpose multilingual data management issues. The commercial database systems [19, 20, 21] provided multilingual support to some extent but not the full functionalities of MDMS. These commercial database systems support Unicode 3.01 standard for storing multilingual data. But main limitation of these systems is that they do not provide any mechanism of querying over multiple languages.

3.0 MULTILINGUAL DATA MANAGEMENT: SYSTEM DESIGN AND ARCHITECTURE

The system architecture for the multilingual data management comprises two main modules: management module and storage module. The management module is responsible for performing inter-language mapping and querying. It consists of two sub modules: query execution and translation. The storage module manages the storage of the encoded database and data dictionaries. The overall system architecture is shown in Fig.1. The single directional arrows represent the direction of next sub-module to be executed in a module whereas bi-directional arrows represent the relationship among the sub-modules for processing. The relationships among the sub-modules are also shown in Fig.1.

3.1 Management Module

Management module consists of two sub-modules: query execution and translation.

3.1.1 Query Execution

The query execution module consists of a group of sub-modules that are related to process a query. These sub-modules are -query manager sub-module, query input / response sub-module, parse query sub-module, search dictionaries sub-module, dictionary-to-DB mapping sub-module, and DB-to-dictionary mapping sub-module as shown in Fig.1. Query manager sub-module performs the task of retrieving the query results to the clients.

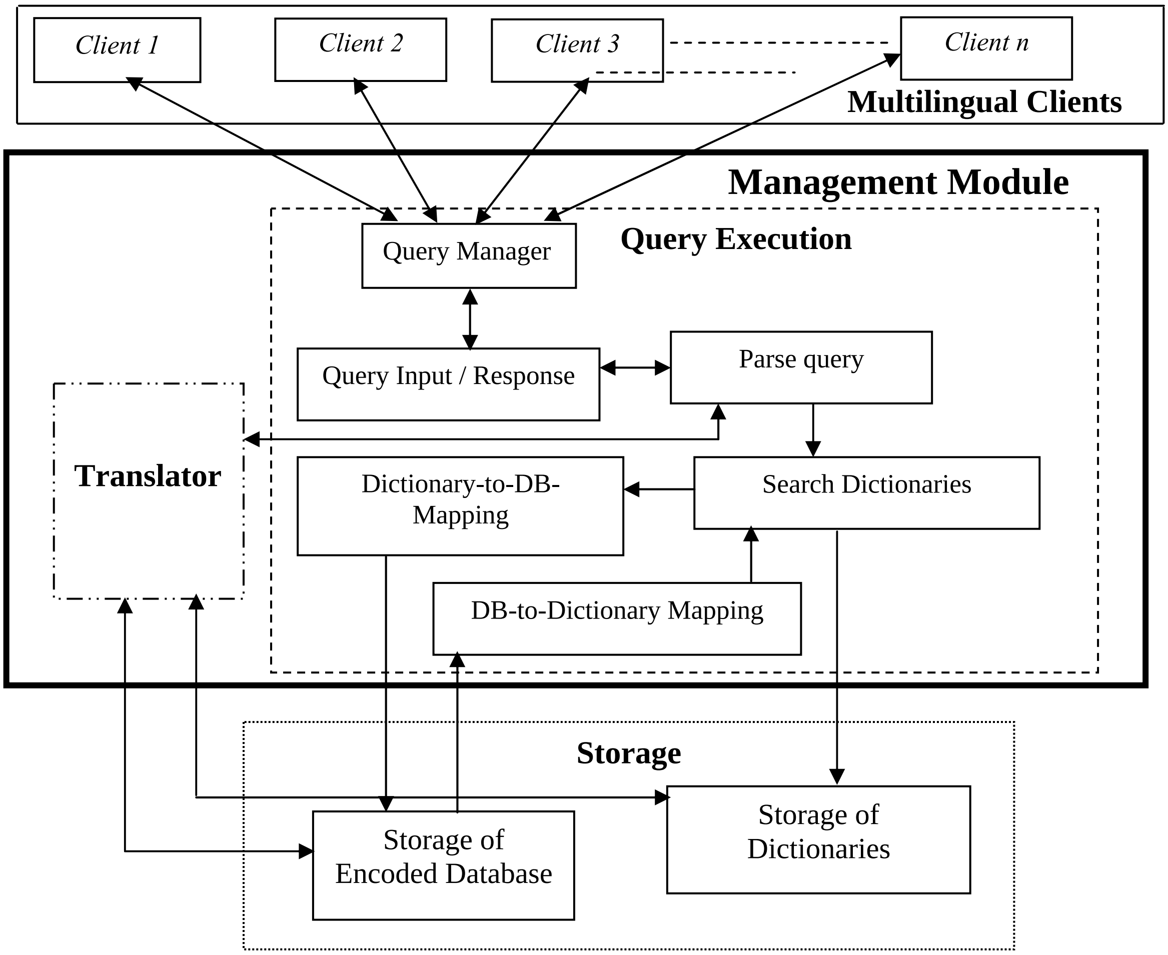


Fig.1: System architecture for multilingual data management

Query input / response sub-module takes the query expression from the query manager and returns the query result to the query manager. Parse query sub-module identifies the important information in the query i.e. for which purpose query is generated. Appropriate dictionaries can be selected, when necessary using search dictionaries sub-module. Dictionary-to-DB-mapping sub-module performs the tasks of linking between data dictionaries and database so that retrieving of information from the database can be done easily. DB-to-dictionary-mapping sub-module is used to perform reverse mapping. This sub-module helps to get the appropriate information for the client.

3.1.2 Translation

In multilingual translation, a record provided in specific language is translated in the target language with the help of a translator. Fig. 2 shows the multilingual translation procedure.

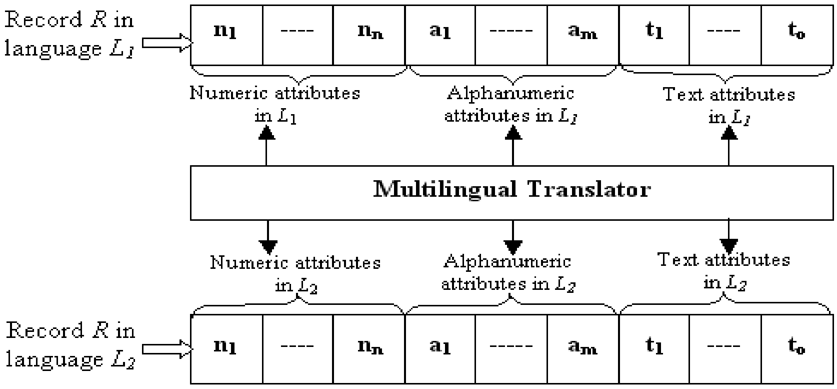

Record *R* in language L_1
n1
----
nn
a1
-----
am
t1
----
to
Numeric attributes in L_1
Alphanumeric attributes in L_1
Text attributes in L_1
Multilingual Translator
Numeric attributes in L_2
Alphanumeric attributes in L_2
Text attributes in L_2
Record *R* in language L_2
n1
----
nn
a1
-----
am
t1
----
to

Fig. 2: Multilingual translation procedure

From the Fig. 2, it is observed that *R* is a record in language L_1, which contains *n* numeric attributes $n_{1,}$ ---, n_n, *a* alphanumeric attributes a_1, --, a_m and *t* text attributes $t_{1,}$ --, t_o. For translating the record *R* in language L_2 we need to encode *R* with the help of a translator. The translation of numeric and alphanumeric and data can be done by providing character-by-character mapping among languages while the translation of text data requires the involvement of data dictionaries. Considering the above factors the algorithm for multilingual translation is as shown below.

```
Algorithm MullTranslator (Record R ){
// R is a record in operation which contains n numeric attributes,
// a alphanumeric attributes and t text attributes. Numeric and alphanumeric
//attributes will be handled directly and text attributes will be handled with the
//help of dictionaries.
for each record R do {
    find numeric attributes n, alphanumeric attributes a and text attributes t
    for n and a call AlphanumericTranslator (n, a);
    for t call Addlexeme (t);
```

From above algorithm, it is observed that *Mulltranslator* algorithm takes a record *R* as its argument and calls *AlphanumericTranslator* for numeric and alphanumeric attributes and *Addlexeme* for text attributes. *AlphanumericTranslator* takes numeric attribute *n* and alphanumeric attribute *a* of record *R* as its argument and performs direct mapping with the database. The algorithm inserts numeric data in binary format and alphanumeric data in ASCII format. The algorithm for translating numeric and alphanumeric data is given below.

```
Algorithm AlphanumericTranslator (n, a ) {
// This algorithm takes numeric attributes n and alphanumeric attributes a of a
//record R as an argument and perform direct mapping with the database.
  for each character c in each value of n or in each value of a in R do
    Search appropriate character wise mapping between database storage
     language and desired language.
```

Addlexeme algorithm takes text attributes *t* of the record *R* as its argument and performs translation among languages with the help of domain dictionaries. The algorithm first searches the corresponding domain dictionary for the existence of the text item in the dictionary. If the item exists in the dictionary the *Addlexeme* algorithm does not insert the item in the dictionary. Otherwise the algorithm inserts the item in the corresponding dictionary with its translated values in other languages.

So, there is only one entry of a specific text item in the domain dictionary. The *Addlexeme,* algorithm for addition of lexeme is given below.

```
Algorithm Addlexeme (text t ) {
// This algorithm takes text attributes t of a record R as an argument and
//perform mapping with the database with the help of codes.
for each t do
   search corresponding dictionary
for each data item x  in each t do
    search for all x in the corresponding dictionary {
       if found (x)
          return corresponding code
      Else
          add x in the dictionary with its translated value in other languages and
           return corresponding code
```

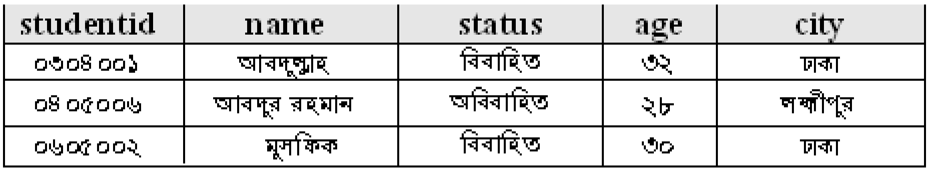
3.2 Storage

Data stored in the database-tables are generally text, numeric or alphanumeric. In practice most of the data is numeric and alphanumeric. Consider the relation *student* with attributes *studentid*, *name*, *status*, *age* and *city*. As an example, consider the storage of student relation in English and the native language Bangla. Table 1 and Table 2 show these two relations.

Table 1: Student_english: Storage of student records in English

studentid	Name	status	age	City
0304001	Abdullah	Married	32	Dhaka
0405006	Abdur Rahman	Unmarried	28	Lakshmipur
0605002	Mushfuk	Married	30	Dhaka

Table 2: Student_bangla: Storage of student records in Bangla



studentid	name	status	age	city
০৩০৪ ০০১	আবদুল্লাহ্	বিবাহিত	৩২	ঢাকা
০৪ ০৫০০৬	আবদুর রহমান	অবিবাহিত	২৮	লক্ষীপুর
০৬০৫ ০০২	মুসফিক	বিবাহিত	৩০	ঢাকা

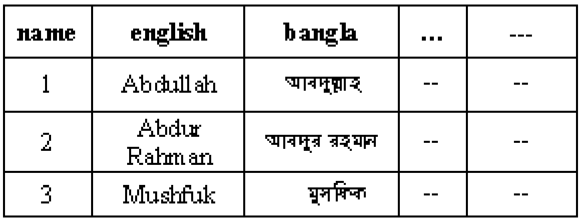
Conventional database systems store relations for each language. So data redundancy is proportional to the number of language support. In the proposed system, a single encoded table is used to store a multilingual table irrespective of the number of languages support. Table 3 is the encoded representation of the tables (Table 1 and Table 2) in English and Bangla.

Table 3: Encoded database storing multilingual data

studentid	name	status	age	city
0304001	1 (01)	1 (0)	32	1 (01)
0405006	2 (10)	2(1)	28	2 (10)
0605002	3 (11)	1 (0)	30	1 (01)

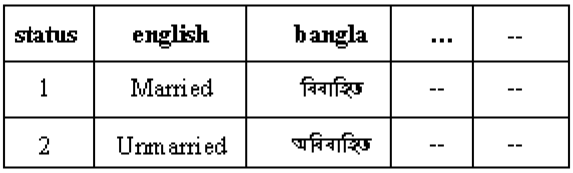
To create the encoded representation, we have three considerations: all numeric fields are represented directly in binary format; all alphanumeric fields are represented in character data and text fields are represented in dictionary encoding method. In Table 1 and Table 2, *age* is a numeric field that has the binary representation with minimum number of bits in the encoded table (Table 3). *StudentID* is an alphanumeric field that has been represented as ASCII character data. The other fields e.g. *name*, *status* and *city* are text fields and encoded using the corresponding domain dictionaries as shown in Table 4, 5 and 6.

Table 4: Dictionary for *name* attribute



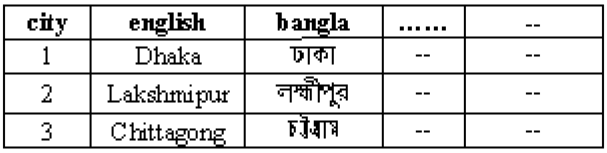
name	english	bangla	...	---
1	Abdullah	আবদুল্লাহ	--	--
2	Abdur Rahman	আবদুর রহমান	--	--
3	Mushfuk	মুসফিক	--	--

Table 5: Dictionary for *status* attribute



status	english	bangla	...	--
1	Married	বিবাহিত	--	--
2	Unmarried	অবিবাহিত	--	--

Table 6: Dictionary for *city* attribute



city	english	bangla	......	--
1	Dhaka	ঢাকা	--	--
2	Lakshmipur	লক্ষীপুর	--	--
3	Chittagong	চট্টগ্রাম	--	--

The dictionaries are created by storing only the unique values of the corresponding domain. The property of the dictionary is that if any *lexeme* is stored in the dictionary, it returns a unique code and vice versa i.e. *code ← encode (lexeme)* and *lexeme← decode (code*). Table 4 shows the dictionary of *name* attribute with *language 1* as English and *language 2* as Bangla. There are three name instances in English and the corresponding values in Bangla in the following column.

So, in the name field of the encoded table code 1 will represent Abdullah or আবদুল্লাহ . Similarly, Abdur Rahman or আবদুর রহমান and Mushfik or মুসফিক will be represented using code 2 and 3 respectively in the encoded table (Table 3). Similarly, Table 5 and Table 6 show the dictionaries for *status* and *city* attributes respectively. The code corresponding to a data item stored in different languages (here in English and Bangla) will represent that data in the encoded table. So, the storage in the encoded table is independent of the number of language support.

3.3 Operations in Multilingual Database

Different operations such as multilingual query, insertion, deletion and update can be performed in the developed system efficiently. The following subsections describe the procedures for different multilingual operations.

3.3.1 Analysis of Multilingual Query Procedure

Multilingual query should be executed efficiently so that correct tuples are retrieved from the database. The algorithm for multilingual query processing is given below.

```
Algorithm MulQuery (A, B) {
// A and B are the set of attributes involve in query execution and query response
// respectively. Both A and B can contain numeric, alphanumeric and text attributes.
    for each A do
    {
        find numeric attributes $A_n$, alphanumeric attributes $A_a$ and text attributes $A_t$ from A
        for $A_n$ and $A_a$ call AlphanumericTranslator ($A_n$ ,$A_a$ ) to search database.
        for $A_t$ call  Addlexeme ($A_t$ ) and use returned codes to search database.
     }
    for each B do
     {
        find numeric attributes $B_n$, alphanumeric attributes $B_a$ and text attributes $B_t$ from B
         for $B_n$ and $B_a$
            call AlphanumericTranslator ($B_n$,$B_a$) to return numeric and  alphanumeric data
            from the database.
          for $B_t$,
             call Addlexeme ($B_t$) and use codes to return text data from the database.
```

Consider the information of some students has been stored in an information theoretic approach in the multilingual database. Student record includes *student ID, name, status, age* and *city*. *Name*, *status* and *city* of students have been stored using dictionary encoding method and *student ID* and *age* have been stored directly in the encoded DB. Now for retrieving students' records living in Dhaka city in English by a client the following is the query to be executed.

```
select st.sudenttid,st.age, N.english, S.english ,C.english
from student as st, Dname as  N, Dstatus as S, Dcity as C
where C.english = 'Dhaka' and st.city = C.city and st.name = N.name
and st.status = S.status
```

In the above query, *Dname*, *Dstatus* and *Dcity* are the domain dictionaries for *name*, *status* and *city* and *student* is the name of the encoded table. For executing the above query, algorithm *MulQuery,* first searches the dictionary *Dcity* for *city* attribute by calling *Addlexeme* algorithm. *Addlexeme* algorithm returns the code corresponding to *Dhaka*. This code will be checked in the *city* field of the encoded table *student*. Then all the matching records corresponding to the code for *Dhaka* will be retrieved from the database. After retrieving the records from the database, the text data i.e. *name*, *status* and *city* will be returned using DB-to-dictionary mapping and the numeric and alphanumeric data *studentid* and *age* will be returned directly to the client using *AlphanumericTranslator* in English.

3.3.2 Analysis of Multilingual Insertion Procedure

In multilingual insertion, care has to be taken for handling the dictionaries properly. Otherwise, data redundancy and inconsistency in the dictionary can be occurred. Also mapping among data dictionaries and database must be handled carefully for removing difficulties in accessing data from the encoded database. The algorithm for multilingual insertion is given below.

```
Algorithm MulInsertion (R){
// R is the record to be inserted which contains $R_n$ numeric, $R_a$ alphanumeric
 // and $R_t$ text attributes.
 {
     for each R find $R_n$, $R_a$ and $R_t$
     for all $R_n$ and $R_a$ call AlphanumericTranslator ($R_n$,$R_a$ )
     for all $R_t$ call Addlexeme ($R_t$ )
```

Now, consider that a client wants to insert the record shown in Table 7 in English.

Table 7: Record to be inserted

StudentID	Name	Status	Age	City
0706005	Rahim	Unmarried	35	Lakshmipur

For inserting the record shown in Table 7, the client has to execute the following SQL statements.

```
insert into student (studentid, name, status, age, city )
select ‘0706005’, N.name, S.status ,35, C.city
from Dname as  N, Dstatus as S, Dcity as C
where N.english = ‘Rahim’ and S.status = ‘unmarried’
        and C.city = ‘Lakshmipur’
```

For inserting the record shown in Table 7, *MulInsertion* algorithm first identifies *studentid* as alphanumeric attribute, *age* as numeric attribute, *name*, *status* and *city* as text attributes. As *studentid* is an alphanumeric value and *age* is a numeric value they will be directly stored in the database with the help of *AlphanumericTranslator*. As *name*, *status* and *city* are text attributes the dictionaries *Dname*, *Dstatus* and *Dcity* corresponding to *name*, *status* and *city* attributes will be searched first. As dictionary *Dname* does not contain the name *Rahim*, *Rahim* will be inserted in the dictionary with its translated value in other languages. A code (here code 5) will be generated at the time of insertion, which will be used to represent *Rahim* in the database. But dictionaries *Dstatus* and *Dcity* contain the value *unmarried* and *Lakshmipur* respectively. So, status and city information will not be inserted in the dictionary. We just pick the codes corresponding to *status* and *city* i.e. code 2 and 2 respectively from dictionaries *Dstatus* and *Dcity* respectively. So, the record will be represented in the encoded table as shown in Table 8.

Table 8: Record in the encoded table

StudentId	Name	Status	Age	City
0706005	5	2	35	2

3.3.3 Analysis of Multilingual Deletion Procedure

To delete records from the database we will have to search the records first. The algorithm shown below performs the task of multilingual deletion.

```
Algorithm MulDeletion (R) {
//Delete record R, which contains numeric, alphanumeric and text attributes.
    Call MulQuery (A, B) to search for R
    check the constraints applied and perform deletion from the database not from
    the dictionary.
```

From the above algorithm, it has been observed that deletion will be performed from the encoded database, not from the dictionary. This is because dictionary information might be needed in future for different types of multilingual operations.

3.3.4 Analysis of Multilingual Update Procedure

Multilingual update procedure involves querying the records. After retrieving the records update operation will be performed. The algorithm for multilingual update is given below.

```
Algorithm MulUpdate (R) {
// R is the record to be updated which contains numeric, alphanumeric
// and text attributes.
 for each  R, call MulQuery (A, B) for retrieving R
     if the value to be updated is text then
        call Addlexeme (t)
if the value to be updated is numeric and alphanumeric then
     call AlphanumericTranslator (n, a )
```

Multilingual update operation is more time consuming than other operations. The *MulUpdate* algorithm first checks the existence of the text data in the corresponding domain dictionaries with which replacement will be made. This is because if the data in the dictionary further insertion of this data in the dictionary will create data duplication. But if it is not in the dictionary we have to insert it in the dictionary with its translated value and code. This code will replace the corresponding code in the compressed database. As in translator-based approach numeric and alphanumeric data is directly inserted in the database, we will replace the numeric and alphanumeric value directly without the involvement of dictionaries.

3.3.5 Use of Two-Phase Commit Protocol

Two ensure the integrity of data proposed multilingual system follows the ACID properties of the transactions. ACID properties state that all the sites in which a transaction *T* executed must agree on the final outcome of the execution and execution of a transaction *T* in isolation preserves the consistency of the database. *T* must commit at all sites or it must abort at all sites. To ensure that our system always follows ACID properties we have used two-phase commit protocol. In this proposed multilingual system, the dictionaries and encoded DB will always be consistent after the execution of any transaction at any site.

3.3.6 Mapping between Database and Data Dictionaries

For efficient handling of multilingual data, proper mapping between dictionaries and database is a crucial factor. Forward mapping or dictionary-to-DB mapping is needed when we want to store something in the database. Forward mapping should be handled in such a way that there will be no ambiguity in the database or dictionaries. For this purpose, care has to be taken so that appropriate code is selected and it is placed in the appropriate field of the database. On the other hand, backward mapping of DB-to-dictionary mapping is needed at the time of retrieving information. In backward mapping selection of appropriate values in appropriate language from the dictionaries based on codes should be done accurately and efficiently.

3.3.7 Complexity Analysis

The multilingual data management system developed has two parts for the time concern; one for searching and storing the necessary information in the dictionary and another is for dictionary and database mapping at the time of different operations. For complexity analyses of the developed multilingual system consider the following.
Size of each block so that the block entirely fits in memory = B_s
Number of blocks needed to store each of n dictionaries = D_b
Number of blocks needed to store encoded table = E_b
Hash function that maps each tuple of a dictionary in any of D_b blocks = h
Let the multilingual operation is based on N attributes that consists of S text attributes and remaining numeric and alphanumeric attributes. Therefore, the complexity can be calculated as follows:

Worst case:
Worst case query complexity occurs when all the attributes involved in query are text attributes and values in the encoded table are in different blocks.
So, in such situation maximum block transfer corresponding to a query = $N + E_b$
Here, E_b is the total number of block transfer of the encoded table.

Best case:
Best case query complexity occurs when all the attributes involved in query are numeric / alphanumeric attributes and values in the encoded table is in the same block.
So, in such situation maximum block transfer corresponding to a query = 1

Average case:
Average case query complexity occurs when number of attributes involved in query are both numeric/alphanumeric and text attributes and values in the encoded table is obtained by accessing fewer blocks. So, in such cases maximum block transfer corresponding to a query that includes S text attributes = $S + E_f$. Here, E_f means number of block transfer corresponding to encoded table.

4.0 RESULTS AND DISCUSSIONS

This section discusses the experimental setup for the simulation and the corresponding results.

4.1 Experimental Setup

The multilingual data management system has been developed on a machine having the operating system Windows XP, 2.4 GHz Pentium IV Processor with 512 MB memory. The system was implemented in JDK 1.6 in the front and in Oracle 9i DBMS in the back-end for storing the database, data dictionaries and related information.

For measuring the performance of the proposed multilingual system, we have considered two schemas *student* schema and *library* schema. A data generation program has generated data items of *student* schema. Hundred thousand records were randomly generated for different students under departments of public universities. *Library* schema is chosen from the central library of Chittagong University of Engineering and Technology (CUET), Bangladesh. This schema is used in CUET for maintaining books information in English. Data items of *library* schema are real data.

4.2 Space Requirement Calculation

The developed system has been implemented with six single dictionaries and one encoded table for each of the *student* and *library* schema. The dictionaries for *student* schema are for storing name, status, city, faculty name, department name and university name and the dictionaries for *library* schema are for storing book title, book author, book type, edition place, publisher and edition source. Each of these dictionaries has three fields for storing information in English, Bangla and auto incremented code for mapping dictionary to encoded DB.

In the proposed system, storage is needed for storing text data in the dictionaries. Also storage is needed for storing code corresponding to text data and numeric/alphanumeric data in the encoded table. On the other hand storage of existing systems includes the storage of information separately in each language.

Table 9: Comparative storage requirement of the developed systems and existing systems

No. of records to store	Student Schema		Library Schema	
	Storage requirement in our system in MB	Storage requirement in existing systems in MB	Storage requirement in our system in MB	Storage requirement in existing systems in MB
1000	0.484	0.801	0.689	1.0872
5000	1.989	4.005	3.086	5.436
10000	3.01	8.01	3.563	10.872
20000	4.721	16.022	4.516	21.744
40000	7.813	32.042	6.424	43.488
60000	10.901	48.065	8.331	65.232
80000	13.991	64.087	10.238	86.976
100000	15.631	81.108	12.146	108.72
200000	23.833	160.218	21.682	217.44

The storage requirement for different number of records shown in Table 9 for existing systems is obtained by summing up the storage required in English and Bangla. Using the data of Table 9, we have obtained the graph as shown in Fig. 3. The graph in Fig. 3 shows the comparative storage requirement for *student* schema and *library* schema.

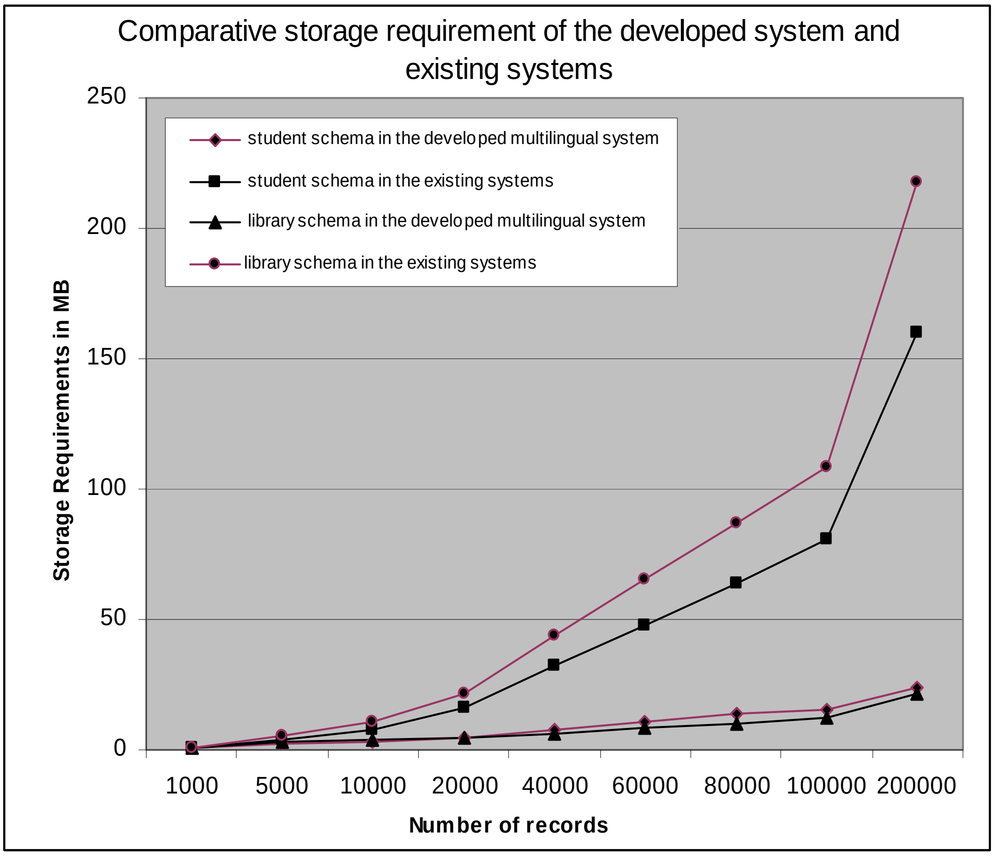


Fig. 3: Comparative storage requirement of the developed multilingual system

From Fig 3, it is observed that the rate of increase of storage requirement with the increase of number of records is significantly higher in existing systems compared to our system. This is because in the our multilingual system when dictionaries are converged there is no further entry in the dictionary and the storage of the compressed database is independent of the number of language support. But in existing systems it is necessary to insert records separately in each language.

4.3 Multilingual Translation Performance

For measuring the efficiency of the translator, we have applied the developed translator on different numeric, alphanumeric and text data of *student* schema and *library* schema. We have randomly chosen one thousand records of both *student* schema and *library* schema and found the translation accuracy of the proposed multilingual system. We have found the accuracy of the translator as shown in Table 10.

From the information of Table 10, we have found that the developed translator performs accurate translation of numeric and alphanumeric data and sufficient translation of text data. Also it is observed that translation of text items in case of real data set is better than random generated data set.

Table 10: Translation accuracy of the developed translator

No. of records	Student schema		Library schema	
	Text translation accuracy (%)	Numeric / alphanumeric translation accuracy (%)	Text translation accuracy (%)	Numeric / alphanumeric translation accuracy (%)
100	76	100	73	100
200	71	100	76	100
300	73	100	75	100
400	68	100	73	100
500	77	100	78	100
600	74	100	83	100
700	71	100	88	100
800	75	100	76	100
900	72	100	80	100
1000	76	100	82	100

This is because in real data set all the entries are valid and understandable. From our experiment we have obtained the result corresponding to translation time as shown in Fig. 4.

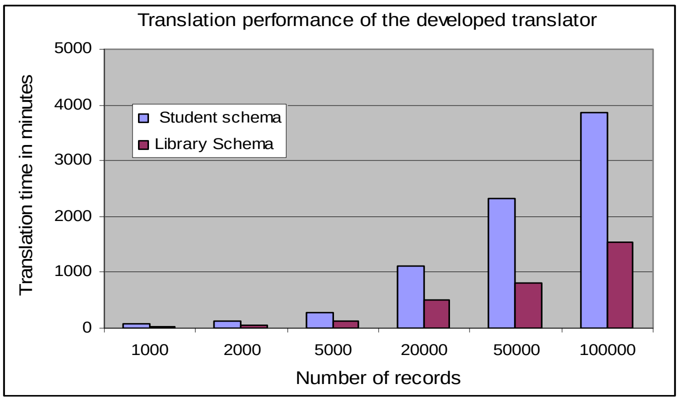


Fig. 4: Translation time requirement of the developed translator

4.4 Multilingual Query Performance

Query performance in our MDMS has been measured by combining the dictionary search time and database search time. For getting accurate performance, we have generated same query on different size of the database and different queries on same dataset. In our experiment, we have considered ten different criterions for performing query operation, which are given below:

- Q1: Query based on text fields and returned values are also text fields.
- Q2: Query based on numeric/alphanumeric fields and returned values are also numeric / alphanumeric fields.
- Q3: Query based on text fields and returned values are numeric/alphanumeric fields.
- Q4: Query based on numeric/alphanumeric fields and returned values are text fields.
- Q5: Query based on both text and numeric/alphanumeric fields and returned values are text fields.

- Q6: Query based on both text and numeric/alphanumeric fields and returned values are numeric/alphanumeric fields.
- Q7: Query based on text fields and returned values are text and numeric/alphanumeric fields.
- Q8: Query based on numeric/alphanumeric fields and returned values are text and numeric/alphanumeric fields.
- Q9: Query based on both text and numeric/alphanumeric fields and returned values are text and numeric/alphanumeric fields with less text attributes than numeric/alphanumeric attributes involved in query.
- Q10: Query based on both text and numeric/alphanumeric fields and returned values are text and numeric/alphanumeric fields with more text attributes than numeric/alphanumeric attributes involved in query.

We have considered different text and numeric/alphanumeric fields for the experiment in both *student* schema and *library* schema. Using the above query conditions in both English and Bangla on different database size and averaging the result obtained in each language, we have obtained the query time for both student schema and library schema as shown in Table 11 and Table 12

Table 11: Query time in milliseconds on *student* schema

No. of records	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**	**Average query time**
1000	2458	230	1163	1418	2775	1066.5	2246.5	1320.5	1780.5	2358	1682
5000	3914	273	2529	2952.5	4326.5	2265	3523.5	2393.5	3768.5	4132	3008
10000	4558.5	375	3515	3974	5278	2854.5	4804.5	3169.5	4314.5	4643.5	3749
20000	6661.5	723.5	4792	5182.5	6952	3671	5786.5	4326.5	6157.5	6541.5	5079
40000	8486.5	1159	5603	5935.5	8731	4495	6582	4913	7245.5	7877	6103
60000	9397	1420	6603	7025.5	9287.5	5304.5	7952.5	5652.5	8378	8742	6976
80000	10825.5	1601	7369	7979	12152.5	6044.5	8822	6446.5	9980	10427.5	8165
100000	12200.5	1777	7974.5	8440	13707.5	6899	9734.5	7502	10841	11334	9041

Table 12: Average query time in milliseconds on *library* schema

No. of records	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**	**Average query time**
1000	2287	231	1100	1383.5	2686	940.5	2177.5	1279	1704.5	2265	1605
5000	3788	278	2483	2880.5	4317.5	2255.5	3530	2343	3706	4089.5	2967
10000	4529	392.5	3569.5	4000	5205	2737.5	4737	3172.5	4289	4246	3688
20000	6631	705.5	4871.5	5093.5	6862	3666	5776	4232	6050.5	6516.5	5040
40000	8369.5	1160	5671.5	6017	8794	4451.5	6603.5	4890	7325	7969.5	6125
60000	9449	1410	6660.5	7124	9276.5	5382	7864	5704.5	8358.5	8715	6994

80000	10704.5	1491.5	7420.5	8077	12375	6185	8889.5	6366.5	10088	10266	8186
100000	12317.5	1841.5	8172	8321	13669.5	6852	9525	7469	10660	10741.5	8957

For comparative performance analysis of multilingual query of the developed MDMS, we have first calculated the average query time as shown in Table 11 and Table 12 and compared this time with the time obtained using same queries (used earlier in the developed MDMS) separately in English and Bangla database on same record size (as used in the developed MDMS) and then averaging the query result obtained separately in English and Bangla. From the experiment, we have obtained the comparative query performance of the developed MDMS and existing systems as shown in Fig. 5.

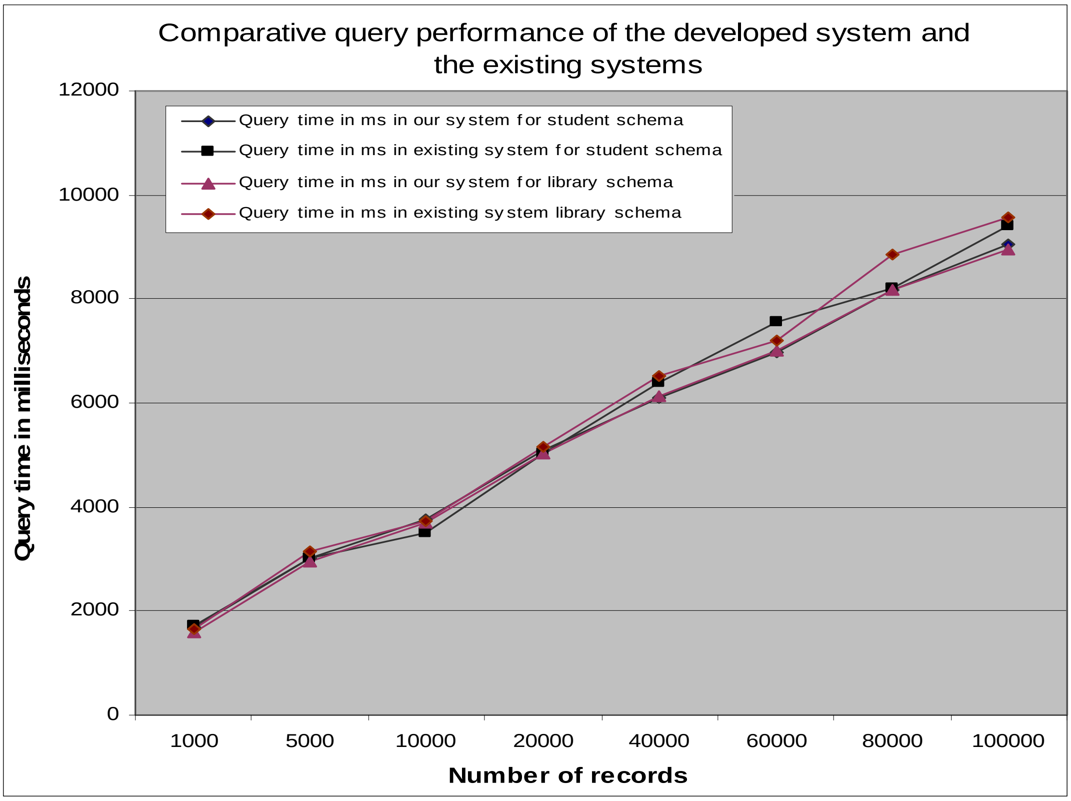


Fig. 5: Comparative multilingual query performance

From Fig. 5, it is observed that multilingual query performance of the developed system is slightly better than the query time of the existing systems. This is because in the developed multilingual system, we need to perform query operation in a single encoded database and the query operation is independent of language selection. But in existing systems we have to use redundant record sets and query operation need to be performed separately from language dependent databases separately to retrieve same data in different languages.

4.5 Multilingual Insertion Performance

For the experimental purpose of insertion, we have considered the matter of random record generation. In this approach when we insert a record it will insert eight text fields, which includes name, father’s name, mother’s name, city, marital status, department, faculty and university and seven numeric and alphanumeric fields, which includes student id, course id, level, term, average marks obtained in a specific term and merit position. From our experiment we have obtained the graph shown in Fig. 6.

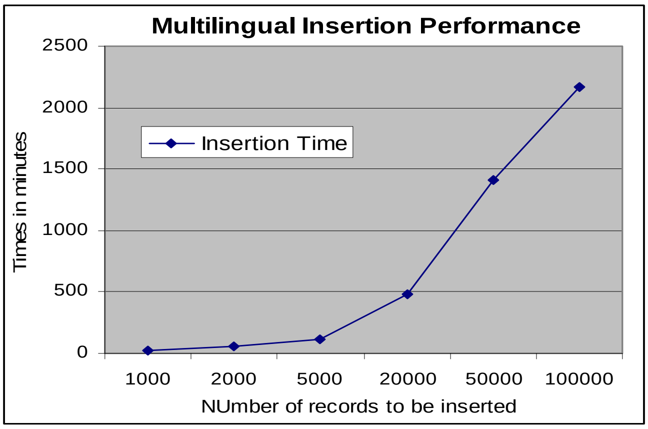


Fig. 6: Multilingual insertion performance

From the graph shown in Fig. 6, it can be considered that if the number of text fields to be inserted increases the insertion time also increases.

4.6 Multilingual Deletion Performance

For the deletion we have to search the record first. In the experiment we have search the records to be deleted using numeric/alphanumeric attributes, as searching using numeric/alphanumeric attributes require less time. From the experiment, we have found the information as shown in Fig.7 while performing multilingual deletion operation.

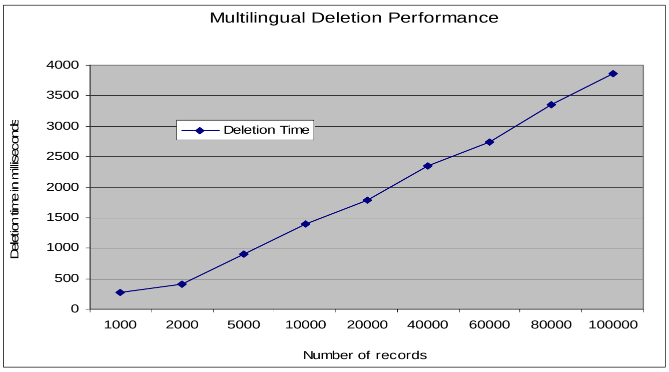


Fig. 7: Multilingual deletion performance

From the Fig. 7, it has been observed that deletion is not so much time consuming with the variation of records. This is because we performed search for tuples to be deleted based on numeric/alphanumeric attributes and deleted tuples directly from the database without involvement of dictionaries.

4.7 Multilingual Update Performance

Update performance is measured by summing up the time required to perform query operation to retrieve the data to be updated and the time required to insert the new data. In our developed MDMS, we have retrieved data to be updated using query based on *studentid* for *student* schema and *subjectcode* for *library* schema. For measuring update performance in the developed

MDMS, we have generated same update operation on different size of the data set in both English and Bangla.

In our experiment, we have considered five different conditions for performing update operation, which are given below.

- U1: Updated values are text data already in the dictionary.
- U2: Updated values are text data not in the dictionary.
- U3: Updated values are numeric/alphanumeric values.
- U4: Updated values involve text data and numeric/alphanumeric data with text data in the dictionary.
- U5: Updated values involve text data and numeric/alphanumeric data with text data not in the dictionary.

We have also considered different fields to be updated. Considering conditions and number of fields to be updated, we have obtained the result as shown in Table 13.

Table 13**:** Update time in milliseconds on different number of records in *student* schema and *library* schema

No. of Records	**Student Schema**						**Library Schema**					
	U1	**U2**	**U3**	**U4**	**U5**	**Average update time**	**U1**	**U2**	**U3**	**U4**	**U5**	**Average update time**
1000	1695	2391	448	1439	2162	1627	1524	2278	403	1384	2012	1520
5000	3915	4781	815	3050	4461	3404	3819	4562	709	2835	4528	3290
10000	5481	6335	1068	4637	5987	4701	5423	6412	958	4682	5944	4683
20000	7126	8036	1197	6138	7824	6064	6981	7836	1120	6045	7819	5960
40000	8235	9148	1352	7935	9058	7145	8151	9125	1273	7889	9025	7092
60000	9227	11520	1473	8944	10869	8406	9113	10521	1365	8812	10726	8107
80000	11243	12782	2268	10543	12065	9780	10983	12659	2178	10223	11954	9599
100000	12198	14936	2504	11087	14684	11081	11025	14561	2518	11034	14869	10801

From Table 13, it is observed that in the developed MDMS maximum update time required when updated values is not in the dictionaries and minimum update time required when update is performed on numeric or alphanumeric data. This is because if updated values are not in the dictionaries, we have to insert the items in the dictionaries first and then we have to make replacement in the encoded DB with the new generated codes. On the other hand if the updated values are numeric or alphanumeric data, we can update the values directly from the encoded DB without the involvement of dictionaries.

For comparative update performance analysis we have done the following:

- Calculated the average update time considering different update conditions on same number of records (as shown in Table 13).
- Calculated the average update time on different size of records considering different update conditions.

- Compared the time obtained in the developed MDMS with the time obtained by performing same update operation separately in English and Bangla database on the same record size and the summing up the update result obtained in English and Bangla.

From our experiment we have obtained comparative update performance as shown in Fig. 8.

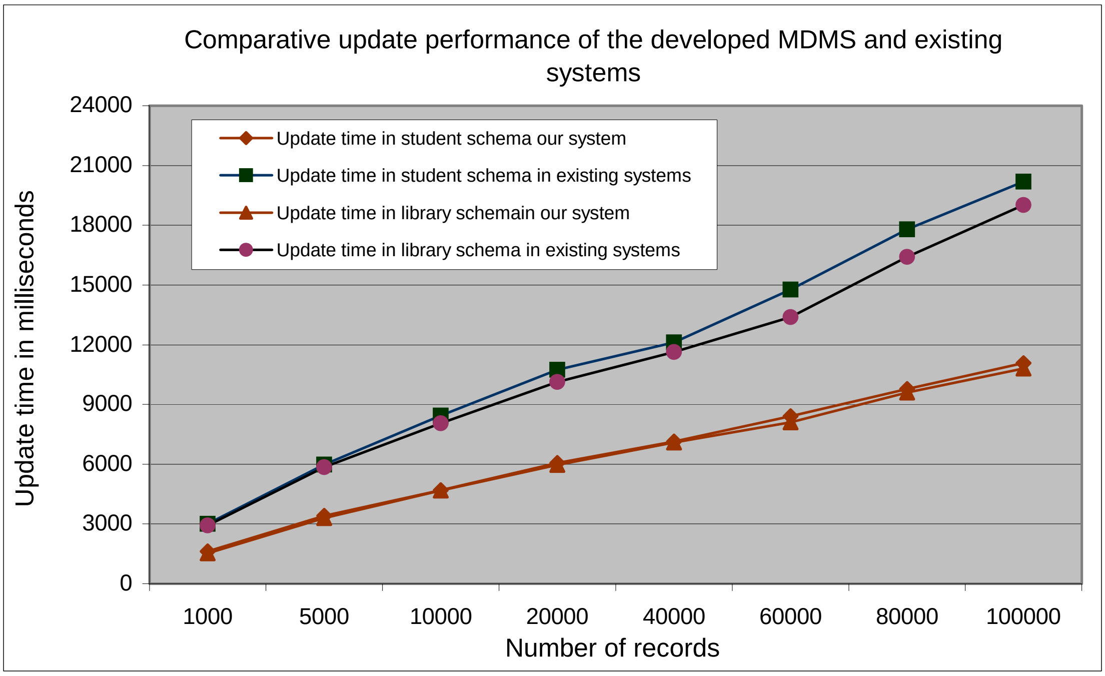


Fig. 8: Comparative update performance of the developed system

From Fig. 8, it has been observed that the developed MDMS required less update time for any update operation compared with the update operations performed in the existing systems. This is because in our system we need to perform update operation on a single encoded database. But in existing systems we have to use redundant databases and update operation need to be performed in each of the databases separately for keeping consistency of information stored in different languages.

5. CONCLUSION

Storage of multilingual data and support of dynamic update is a problem for conventional Database Management Systems. While existing database systems provide some means of storing and querying multilingual data, they suffer from redundancy proportional to the number of language support. In this paper, we have proposed a translator-based approach for handling multilingual data that stores data in information theoretic way with minimum redundancy. We have developed algorithms for insertion of multilingual data into a single non-redundant database, querying and update in the database. The algorithms have been evaluated by syntactic data sets generated by a data generation program and real data sets as well. We have compared the performance of our system with the existing systems. Our system outperforms the existing multilingual systems in terms of both space and time.

We have implemented the system for two languages: English and Bangla. In this system, data has been stored in a central server and clients can perform different operations in the database dynamically in a distributed environment. Schema evolution in the developed MDMS is simple and easier to maintain the database consistency because of a single encoded database. These tasks are difficult in the existing systems due to redundant databases. Query performance in the proposed system is more efficient than the existing systems. Though we have considered Bangla and English languages for the experimental purpose, the system can adapt other languages with the addition of the corresponding language translator only.

REFERENCES

[1] C. Ordonez, “Optimizing Recursive Queries in SQL”, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp 834-839, 2005.

[2] W. Chung, Y. Zhang, Z. Huang, G. Wang, T. H. Ong and T. H. Chen, “Internet Searching and Browsing in a Multilingual World: An Experiment on the Chinese Business Intelligence Portal ”, *Journal of the American Society for Information Science and Technology,* Vol. 55, No. 9, pp 818-831, 2004.

[3] A. Kumaran, P. Chowdary and J. Haritsa “On pushing multilingual query operators into relational engines”, *Proceedings of 22nd IEEE International Conference on Data Engineering (ICDE)*, Atlanta, USA, April 2006.

[4] F. Gey, A. Chen, M. Buckland and R. Larson, “Translingual vocabulary mapping for multilingual information access”, *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval,* Tampere, Finland, pp 455-456, August 2002.

[5] J. Marlow, P. Clough, J. C. Recuero and J. Artiles, “Exploring the Effects of Language Skills on Multilingual Web Search”, *In Proceedings of the 30th European Conference on IR Research (ECIR'08),* Glasgow, UK, April 2008.

[6] The Computer Scope Ltd., Dublin, Ireland. http://www.NUA.ie/Surveys

[7] A. Kumaran and J. R. Haritsa, “On the Costs of Multilingualism in Database Systems”, *Proceedings of 29th Very Large Databases Conference*, Berlin, Germany, pp 105-116, September 2003.

[8] A. Kumaran,., “Multilingual information processing on relational database architectures”, *Ph.D Thesis, Department of Computer Science and Automation*, Indian Institute of Science, Bangalore, India, December 2005.

[9] A. Silberschatz, H. F. Korth, and S. Sudarshan, “Database systems concepts”, *Fourth Edition,* Tata McGraw Hill, pp 709 – 730, 2002.

[10] The Association for Computational Linguistics. http://www.aclweb.org

[11] Special Interest Group in Information Retrieval (ACM SIGIR). http://www.acm.org/sigir.

[12] P. Pingali and V. Varma, “Multi-lingual Indexing Support for CLIR using Language Modeling”, *Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society,* Vol. 30, No. 1, pp 57-72, March 2007

[13] R. King and A. Morfeq, “Bayan: An Arabic Text Database Management System”, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, Vol. 19, Issue. 2, pp 12-23, 1990.

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

[15] M. Andersson, Y. Dupont, S. Spaccapietra, K. Yetongnon, M. Tresch and H. Ye, “FEMUS: A Federated Multilingual Database System”, *Advanced Database Systems, Springer-Verlag*, Vol. 359, pp 359-380, 1993.

[16] C. Yip, B. Kao and D. Cheung, “A Framework for the Support of Multilingual Computing Environments”, *Technical Report, University of Hong Kong*, 1997.

[17] B. Lambert, K. Chang and S. Lin, “Descriptive Analysis of the Drug Name Lexicon” *Drug Information Journal*, Vol. 35 No. 1, 2001.

[18] The EROS System.http://merovingio.c2rmf.cnrs.fr/eros/eros.xhtml

[19] IBM Corporation, Armonk, New York. http://www.ibm.com

[20] Microsoft Corporation, Redmond, Washington. http://www.microsoft.com.

[21] Oracle Corporation, Redwood Shores, California. http://www.oracle.com.

BIOGRAPHY

Dr. Abu Sayed Md. Latiful Hoque received his PhD in the field of Computer & Information Science from University of Strathclyde, Glasgow, UK in 2003 with Commonwealth Academic Staff Award. He obtained M.Sc. in Computer Science & Engineering and B.Sc. in Electrical & Electronic Engineering from Bangladesh University of Engineering & Technology (BUET) in 1997 and 1986 respectively. He has been working as a faculty member in the Department of Computer Science & Engineering of BUET since 1990 and currently his position is an Associate Professor. He is a Fellow of Institute of Engineers Bangladesh (IEB) and Bangladesh Computer Society. His research interest includes Data Warehouse, Data Mining, Information Retrieval and Compression in DatabaseSystems.

Mohammad Shamsul Arefin received his B.Sc. Engineering in Computer Science and Engineering from Khulna University, Khulna, Bangladesh in 2002, and completed his M.Sc. Engineering in Computer Science & Engineering in 2008 from Bangladesh University of Engineering & Technology (BUET), Bangladesh. He is a member of Institute of Engineers Bangladesh (IEB) and currently working as an Assistant Professor of Chittagong University of Engineering & Technology, Chittagong, in Computer Science & Engineering Department. His research interest includes Data Management, Semantic Web, Machine Learning, Natural Language Processing and Object Oriented System Design.