Customized Approach for Enhancing SIMS Performance by using the Concept of Indexing

A. S. Dhanadhya
Government College of Engineering
Amravati 444 604, Maharashtra, India

P. N. Chatur, PhD
Government College of Engineering
Amravati 444 604, Maharashtra, India

ABSTRACT

Now a days everyone is trying to manage the data digitally on computer rather than maintaining it with paper-pen as because the management of data digitally is very easier than maintaining it with paper and pen which was done early days, this is also getting implemented in the field of education very rapidly, hence huge amount of student information is getting stored because of this digitalization of world and increasing amount of students and institutes. And as the size and complexity of the stored data is increasing very fast it is giving an impact on the performance of the management applications used. That is why storing and retrieving the data from the databases of various universities always requires efficient procedures or algorithms. Indexing is a technique which stores and retrieves the data efficiently from the database by mapping with the particular records. Customized indexing methods becomes requisite so as to retrieve and store data from the database in more efficient way. There is a little more time complexity and more wastage of the space for unallocated records in normal indexing with equal number of alphabets in the primary index. This paper proposes a customized and enhanced approach to search the information from such databases with reduced time complexity and latency time by specializing the concept as well as by using indexing with repetitive frequency so as to form the buckets with equal number of records.

Keywords

Indexing, Buckets, Database, Performance.

1 INTRODUCTION

1.1 Problem Definition

Given a database of a Student Information System of some specific institute. Aim is to design a MIS system with the database which is indexed by using the concept of repetitive indexing and considering all possible cases so as to enhance the performance of the existing system without making any additions in the actual database. It can improve the performance i.e. minimize the data retrieval operation time on the existing tables. It contains some survey based observation of seven colleges [1].

By perception the time complexity can be reduced if you organize the database as per the data as it appears by legitimate analysis. As per the analysis, the data is stored into different buckets before it is stored into the database and the bucket may contain many records under that on single block. Like the database of some institutes (as according to the survey) is organized as a set of twenty six buckets. Which means there are 26 buckets for each and every alphabet of total 26 alphabets. And if there are 2700 students, there will be 26 buckets of approximately 100 records each. But in this kind of indexing, a problem is encountered which is that the frequency of 'X' or 'Q' exposing as a starting alphabet would be very close to zero while that of names starting with 'A' or 'S' will be quite higher.

1.2 Scope of Problem

Currently as you can see Student Information Management System is in practice in each college and university. In such huge management systems, it is very important to worry about the speed of data retrieval. To create databases and index them, there are a lots of soft-wares and tool are available.

So it is being searched for a technique that improves the efficiency of SIMS system than that of the existing one. The country India has a market which is cost sensitive and always looks for a lower cost model for learning management system, also the system should be easy-to maintain, easy-to-use and low-cost.

For this reason, we preferred open-source technologies over the commercial alternatives, concentrated on web-based system so that it can be accessed from anywhere over Internet with any platform using a web browser and have nearly zero requirements for the end-user (no setup).

1.3 Reason for Selecting Problem

As already known and discussed as well, with the growth of time, the information about the students is getting gathered continuously and is getting doubled and redoubled because of increasing number of colleges, universities and the sharply increasing amount of data of earlier and new students. And hence it shows how difficult is the level of management of this very large amount of data. Student Information Management System comes out as a solution for management of this data [2] but again the problem of managing this student data efficiently persists. Hence here coming out with the solution to this problem that is customized indexing, it helps in giving good efficiency for the management of this large data of students through different colleges, institutes, schools. In the existing student information management system you are implementing indexing over the single column only, it increases the information retrieval speed when the query is fired on the table over the column which is already indexed, but doesn't proved the same speed or performance while accessing the data over some other column which is not indexed [3]. Along with this concept it's given a try to provide an extension to it which is use of repetitive frequency while doing indexing.

1.4 Related Work

Given below is the related work done in the field of management of student information.

In 2013, Prabhu T Kannan and Srividya K Bansal, proposed a Unimate: A Student Information System. The goal of this project is to develop a prototype for a low-cost web-based application that provides features of both learning management systems and student information systems, and is customized to the needs of universities in India [4].

TANG Yu-fang and HANG Yong-sheng, in 2009 proposed a paper on "Design and Implementation of College Student Information Management System Based on Web Services" They used .NET framework for the design of this system.

2 SYSTEM DEVELOPMENT

2.1 Aim and Objective

Aim

To enhance the performance of existing student information management system by specialized indexing approach and proper index selection for SIMS and also provide an extension of customized indexing to it.

Objectives

- To gain higher data retrieval speed.
- Modify the level of indexing to better level.
- Do proper index selection for SIMS.

2.2 Research Methodologies Used

INDEXING

Separate data structures that allow DBMS to locate particular records in the file of the base table more quickly are called Indexes. It is not much worry of performance for MYSQL when the tables are small and data size is small. But as the new data gets added and the size of the tables increase, the performance automatically degrades and it may result in slower response time by the application or system. But luckily, indexing in the MYSQL helps in retrieving data records more efficiently and at much better speed. For example, consider the following simple SQL statement.

```
SELECT
AVG (MARKS) FROM STUDENTS
GROUP BY DIVISION;
```

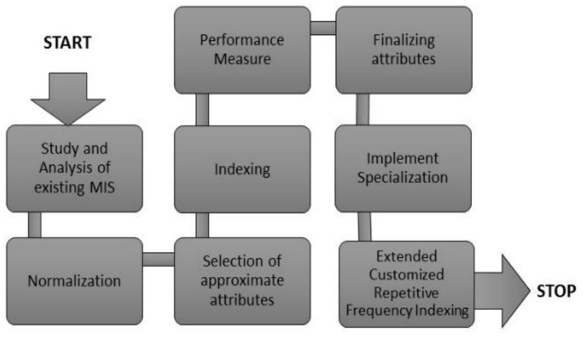

START
Study and Analysis of existing MIS
Normalization
Selection of approximate attributes
Indexing
Performance Measure
Finalizing attributes
Implement Specialization
Extended Customized Repetitive Frequency Indexing
STOP

Figure 1: System Architecture Workflow

In the earlier case, when column MARKS and DIVISION are not indexed, MYSQL will scan the whole table for MARKS. Latter, it scans the whole table to do sorting to group the result by city. The sorting and selecting process becomes a slow one because of this. The MYSQL doesn't need to do a full-length table look-up because index is always sorted (similar to that of an index in an encyclopedia). Therefore, an index on MARKS and DIVISION will increase the process speed.

2.3 Proposed Workflow

Here, use of indexing plays little different role than the existing indexing techniques.

The concept of indexing is specialized for Student Information Management System.

By indexing the column before firing the query it is tried to achieve better execution speed than the speed of query which is fired over the column which is not indexed.

Input: Some College/University/School Student's Database

Output: Modified database which gives more efficiency for fired queries.

1. Study the existing System:

- Check if any flaws in the existing system as a third party user.
- Check if any flaws in the existing system at the administrator level.

2. Choose suitable form of normalization and apply normalization over it.

3. After studying the overall system, choose the columns to be indexed.

4. Index those selected columns

5. Calculate and then compare the performance of those when indexed and when not.

6. Finalize the best selected columns

7. Specialize the existing database for an application of Student Information Management System

8. Provide an extension of customized indexing i.e. indexing with repetitive frequency.

3 EXTENSION OF CUSTOMIZED INDEXING

To achieve higher data retrieval speeds, the technique of indexing is used. As per the analysis, the data is stored into different buckets before it is stored into the database and the bucket may contain many records under that on single block. Like the database of GCOEA (as per the survey) is organized as a set of twenty six buckets. Which means there are 26 buckets for each and every alphabet of total 26 alphabets. But here you encounter a problem of uneven distribution of number of records in among the total 26 buckets.

SPARSE INDEXING

In dense index, there is an index record for every search key value in the database which helps in searching faster. However it requires more space to store the index records. Search key value and a pointer to the actual record on the disk is contained in the index record. If found, indices with the duplicate keys, index points to the lowest search key in each of the block [5].

Let us consider the example of GCOEA's SIMS student database which is a database of nearly 600 students. Now consider (student name, pointer) pair as a sparse index file, you will see the situation like shown in fig 2.

TIME COMPLEXITY CALCULATIOS

Statement: "Display all details of the students with name C"
Corresponding structured query language statement for the above statement will be as follows

SELECT *

FROM STUDENTS

WHERE NAME = "C"

Now when it starts for searching the detail of the student "C", it is found that C is not there in the index file as it is a sparse index and it comes to the result that "C" falls in the range of "A- D" and hence pointer to A is followed and hence by now time complexity becomes 1 Unit as "A" is on first position in the index.

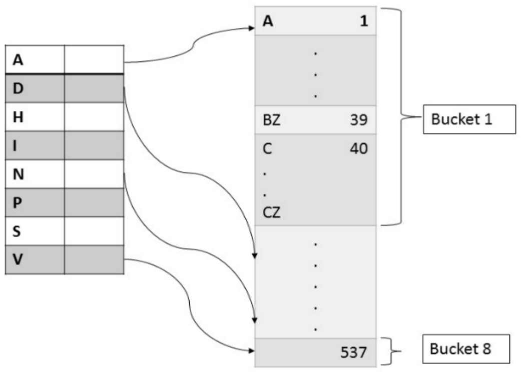


Figure 2: Sparse Indexing Architecture

After following the pointer it starts searching the actual data file, diagram shows as "C" is the 40th record, the time complexity needed to search "C" in actual data file is 40 Units. And hence the total time complexity needed to search "C" becomes 1 + 40 i.e. 41 Units

MULTILEVEL INDEXING

Index records has search-key value and data pointers. Along with the actual database files, the index is stored on the disk. The size of indices grows as the size of database grows. So as to speed up the search, it is important to keep the index records in the main memory. Outer index are the pointers which point to the index file and inner index is the index file that points to the actual data [6]. This architecture is show in fig 3.

TIME COMPLEXITY CALCULATIOS

Again calculate the time complexity on the same GCOEA's SIMS student database with the same SQL statement but the database which is indexed using multilevel indexing.

SELECT *

FROM STUDENTS

WHERE NAME = "C";

Now when it starts for searching the detail of the student "C", it is found that C is not there in the outer index and it comes to the result that "C" falls in the range of "A-J" and hence pointer to A is followed and hence by now time complexity becomes 1 Unit as "A" is on first position in the index. Now it will check for "C" in the inner index as it is a multilevel indexing. As "C" does not appear in the inner index, "A" is followed which increases time complexity to 2. After following the pointer it starts searching the actual data file and hence the total time complexity needed to search "C" becomes 2 + 40 i.e. 42 Units

So now there is not much difference in 41 and 42, hence here a new method is proposed.

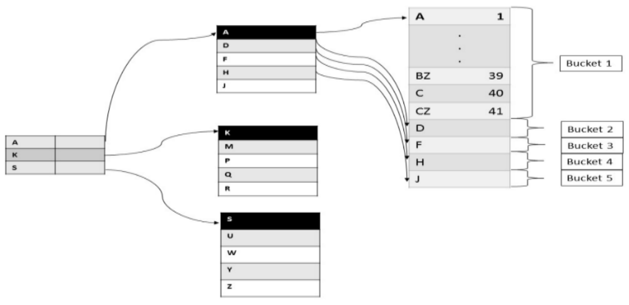


Figure 3: Multilevel Indexing Architecture

PROPOSED METHOD

Before starting the actual proposed method, calculate the percentage of occurrence of each and every alphabet considering it as a starting alphabet in the attribute student name.

Now allocate the buckets as per the calculation of percentage of occurrence and map the pointers. This proposed architecture is shown in the diagram below. Considering the same database of GCOEA's SIMS, the average percentage of occurrence of 'a' as the starting alphabet is 9.2% and then assign two buckets to the records those start with the alphabet 'a'. And similarly as per these percentage values are decided upon the number of buckets to be assigned as per the percentage of occurrence of each alphabet as a starting alphabet.

TIME COMPLEXITY CALCULATIOS

Once again calculate the time complexity on the same GCOEA's SIMS student database with the same SQL statement but the database which is indexed using multilevel indexing.

SELECT *

FROM STUDENTS

WHERE NAME = "C";

Here the way of storing and retrieving the data is little different, calculate the percentage of occurrence for all 26 alphabets and then decide upon some range of percentage for each bucket. It is decided to 7.1 in this method. The table below shows average and percentage of each of these alphabets.

Here in this case the time complexity of searching C is 4 + 1 i.e. 5. As the buckets are allocated depending upon the percentage value of occurrence, the name starting with the alphabet with more frequency percentage is allocated more number of buckets and hence helps in faster searching i.e. reduces the time complexity.

Here in this case the time complexity of searching C is 4 + 1 i.e. 5. As the buckets are allocated depending upon the percentage value of

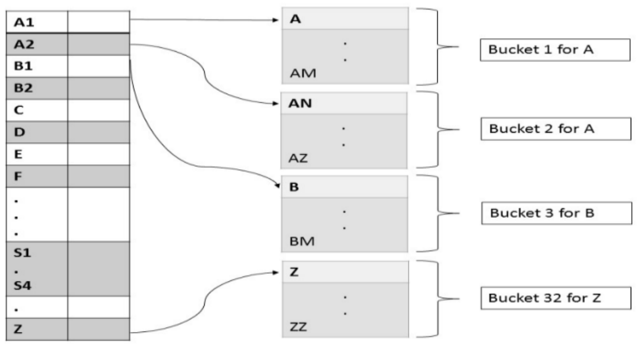
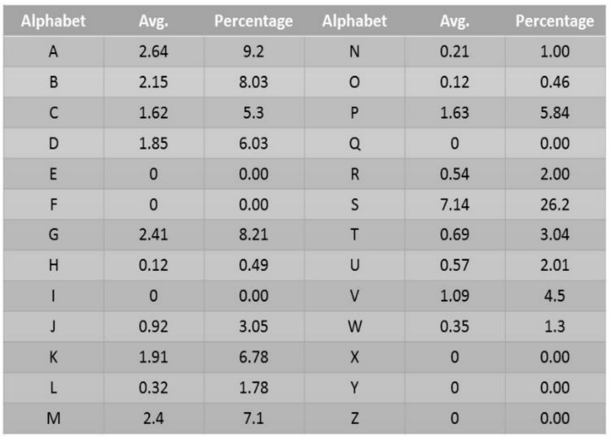


Figure 4: Proposed Indexing Architecture

Table 1: Alphabate occurrence percentage and average



Alphabet	Avg.	Percentage	Alphabet	Avg.	Percentage
A	2.64	9.2	N	0.21	1.00
B	2.15	8.03	O	0.12	0.46
C	1.62	5.3	P	1.63	5.84
D	1.85	6.03	Q	0	0.00
E	0	0.00	R	0.54	2.00
F	0	0.00	S	7.14	26.2
G	2.41	8.21	T	0.69	3.04
H	0.12	0.49	U	0.57	2.01
I	0	0.00	V	1.09	4.5
J	0.92	3.05	W	0.35	1.3
K	1.91	6.78	X	0	0.00
L	0.32	1.78	Y	0	0.00
M	2.4	7.1	Z	0	0.00

occurrence, the name starting with the alphabet with more frequency percentage is allocated more number of buckets and hence helps in faster searching i.e. recuces the time complexity.

4 PERFORMANCE ANALYSIS AND COMPARISON

Consider a test database 'sakila' and next select the table 'sakila.film' which has total 1000 tuples. The schema of the table 'film' is sakila.fillm (title, description, release_year, language_id, original_laguage, rental_duration, rental_rate, length, replacement…). Now fire the following SQL query on it.

SELECT title, description, release_year

FROM sakila.film

WHERE rental_duration = 6;

After running this query, check for the execution time needed in seconds for this query. For this use a tool named "MySQL Query Browser". It gives the query execution time needed for a specific query to run over a specific table of some database. See the results of this in the left hand side of Table 2 below.

Now apply indexing over the column 'rental_duration' and create an index 'tindex' on it and run the same SQL statement again. And note the results of this also which are shown on the right hand side of Table 2.

Once done with gathering the results, calculate average, minimum and maximum query execution time and the query execution time with the highest frequency. So obtain the results as shown above, the average query execution time for existing method is 0.0076 sec while the average query execution time for proposed method is 0.0072 sec. Hence a better performance is achieved with proposed method than the existing one. And you can expect this performance to enhance more as the size of the database will grow more and more.

Table 2: Comparison of query execution time of existing method and proposed method

Query Execution Time (in Seconds) for Existing Method	Query Execution Time (in Seconds) for Proposed Method
0.0079	0.0072
0.0079	0.0072
0.0058	0.0092
0.0077	0.0071
0.0078	0.0072
0.0078	0.0089
0.0080	0.0072
0.0075	0.0075
0.0096	0.0078
0.0078	0.0072
0.0083	0.0071
0.0078	0.0071
0.0081	0.0093
0.0077	0.0066
0.0101	0.0073
0.0069	0.0070
0.0078	0.0081
0.0100	0.0073
0.0076	0.0080

AVG = 0.0076
MIN = 0.0058
MAX = 0.0101
MAX FREQUENCY -> 0.0078 (freq. = 5)
	AVG = 0.0072
	MIN = 0.0066
	MAX = 0.0093
	MAX FREQUENCY -> 0.0072 (freq. = 5)

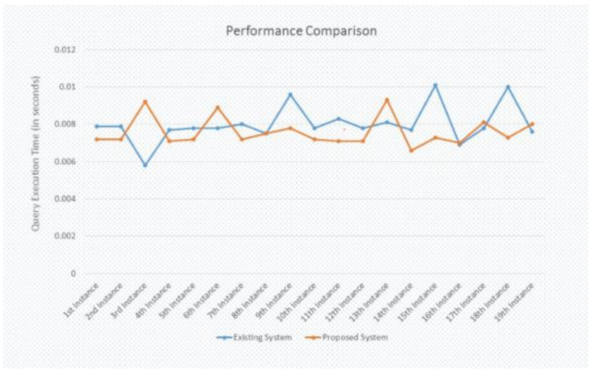


Figure 5: Performance Analysis and Comparison

5 FUTURE WORK AND CONTRIBUTION

5.1 Software Proficiency

MICROSOFT VISUAL STUDIO and MySQL Workbench is used because of its comprehensiveness and ease in using environment for technical outcomes. The high-level GUI and a good user-interface of this project is developed using MICROSOFT VISUAL STUDIO. Also VISUAL STUDIO supports good elements for a web application which gives a very nice and decent look and a friendly environment for the user. At the same time it helps in fast development and execution. A good database server has been provided by MySQL Workbench for managing a university/institute level database. And for performance evaluation use a tool called MySQL Query Browser.

5.2 Contribution in Future Work

- In almost all the institutes, universities or schools the system will be useful.
- A better speed of retrieval can be achieved as another advantage after implementation of this idea of specializing the concept of indexing for SIMS. Hence the most important contribution in future will be in changing the whole Student Information Management System.
- One major thing or future scope is to combine the Student Information Management System with the concept of repetitive frequency indexing which is discussed here.

6. CONCLUSION

It is a challenging problem in the field of computer science to manage the large size data efficiently. The performance analysis discussed above shows that the proposed systems gives a good performance than the existing one. The advantage of equal number of records per bucket is achieved by the customized indexing which is based in the frequency. The comparison shows that the Time complexity has been reduced drastically and hence we achieve the faster search.

7. REFERENCES

[1] Jin Mei-shan, Qiu Chang-li and Li Jing, \The Designment of Student Information Management System Based on B/S Architecture", 2nd International Conference on Consumer Electronics, Communications and Networks (CEC-Net), 2012, pp. 2153-2155.

[2] Prabhu T Kannan and Srividya K Bansal, \Unimate: A Student Information System", IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2013, pp. 1251-1256.

[3] GCOEA MIS System (http://10.121.11.205/GCAMRAVATI/index.html), Access Date: 10 September 2014.

[4] TANG Yu-fang, HANG Yong-sheng, "Design and Implementation of College Student Information Management System Based on Web Services", 978-1-4244-3930-0/09/$25.00 ©2009 IEEE.

[5] A. S. Dhanadhya, Dr. P. N. Chatur, "Enhancing Information Management System Performance by Specializing the Concept of Indexing", IJECS Volume 4 Issue 1 January, 2015.

[6] Prabhu T Kannan, Srividya K Bansal, "Unimate: A Student Information System", 978-1-4673-6217-7/13/$31.00_c 2013 IEEE.

[7] Yaoping Wang ; Vu Pham ; Karmouch, A., "Issues on the design of a global university database system," IEEE International Conference on Computer Design, 2006. ICCD 2006.

[8] Miihleisen, H.; Walther, T.; Tolksdorf, R.; "Multi - level indexing in a distributed self-organized storage system" 2011 IEEE Congress on Evolutionary Computation (CEC).

[9] Yan Cao; He-feng Tong; Jie Yu; Dar-zen Chen; Mu-hsuan Huang; Xu Zhang; Yong Luo; Yun-hua Zhao; Ze-yu Zhang "Performance evaluation of universities in China based on ESI database," 2010 Proceedings of PICMET '10.

[10] Anderson, N.; McMaster, K.; Sambasivam, S "A comparative study of students' conceptual database frameworks across universities," 2010 IEEE Frontiers in Education Conference (FIE).