Scientific Research and Essays Vol. 5(24), pp. 3994-4001, 18 December, 2010
Available online at http://www.academicjournals.org/SRE
ISSN 1992-2248 ©2010 Academic Journals

Full Length Research Paper

An improved architecture design of shared repository database model

Mu'tasam Okasheh[1], Adnan I. Al Rabea[1]* and Ibrahiem M. M. El Emary[2]

[1]Al-Balqa Applied University, Al Salt, Jordan.
[2]King Abdulaziz University, Jeddah, kingdom of Saudi Arabia.

Accepted 24 November, 2010

For many years ago, the term repository has been used in a restricted manner, since it only addressed metadata management in the context of database management systems. Nowadays it is used in a much broader sense and covers multiple services supporting design applications. Our concern is to make well customized repositories available for users. To reach this goal, we present this paper which has a main objective of improving the performance of shared database pattern in the Repository Model. Accordingly, the presented work aims to increase the level of performance of the shared repository pattern when large amounts of data are to be shared by reducing the number of times that is required to access the central database and also to save the cost rather than increasing the speed of response for the subsystems requests to access their desired data. Accordingly, the idea of improvement was based on the establishment of buffer where repository model is an organization model used as a storage medium of a huge database. Also, it is used in interactions between the subsystems of the system in architectural design showing the exchange of data between the subsystems in the system as a whole. The proposed model to achieve the required enhancement was simulated, and the output simulated results show its capability of reducing the access time of data, increasing the speed of accessing the data, increasing the system performance, and also increasing the efficiency of the system work. Also, the proposed model was compared with the approach of shared repository pattern and the results showed that the proposed approach outperforms better through making enhancement in reducing the access time and increasing the speed of accessiing the data.

Key words: Database repository model, buffer, subsystem, RDBMS, central database, shared database.

INTRODUCTION AND MOTIVATION

Two kinds of technology can be found which aim at an adequate support of design applications: frameworks (Shaw et al., 1995; Shaw and Paul, 1997; Somerville, 2007) and repositories (Van der, 1994; Wakeman and Jowett, 1993; www.eric.ed.gov). Although the goals of both are pretty close, we think, there are some differences between the underlying approaches. Frameworks focus on providing basic services, which may be adapted to special application needs, and, thus, help to provide corresponding design environments capable of supporting the (special) design application. Repositories or, more precisely, repository managers, on the other hand, emphasize the data control aspect of a certain class of design applications, e. g. software development applications, and offer predefined, generic services. We think the two approaches may be integrated in a way beneficial to both, system providers and system users. In simple words, we want to provide a framework allowing the generation of repository managers.

In Vander (1994), the term repository is defined as a shared database of information about engineering artifacts. Thus, a common repository allows (design) tools to share information so they can work together. A corresponding repository manager provides services for modeling, retrieving, and managing objects in a

*Corresponding author. E-mail: adnan_alrabea@yahoo.com, omary57@hotmail.com.

repository. For that purpose, a repository manager has to provide the standard amenities of a DBMS (data model, queries, views, integrity control, access control, and transactions) as well as value-added services (Van der, 1994): checkout/check in, version control, configuration control, notification, context management and workflow control. Shared databases are used for knowledge exchange in groups. Whether a person is willing to contribute knowledge to a shared database presents a social dilemma: Each group member saves time and energy by not contributing any information to the database and by using the database only to retrieve information which was contributed by others. But if all people use this strategy, then the database will be empty and, hence, useless for every group member. Based on theoretical approaches, two models for fostering the information-sharing behavior of database users were presented in (www.techgenie.com): one for enhancing the quality of database contents, and one for enhancing the quantity of those contents. The models take into account the following factors: the kinds of rewards the participants obtain for contributing information, the individual costs associated with this contribution, the prospective metaknowledge about the importance of one's own information to the others, and the retrospective metaknowledge about how much others contributed to and retrieved from the database. These factors enhance the quantity of database contents as well as their quality. A highly controlled experimental setting for testing the models is presented. Results of three experiments support some expectations derived from the models.

Dealing with a huge volume of data within a system which contains subsystems is essential for making interaction between subsystems, and shared central database in order to obtain data. This process is needed or required to provide service to the subsystem requesting service from the central database (Philippe, 1998). The importance of this paper is regarding the data repository model and improving the shared database return to the benefiting of each subsystem in getting the service so as to address the problems encountered when somebody requests for data from the central database using one or all of the subsystems. Here, if the first subsystem asked for data from the central database and got it, and after that another subsystem asked for the same data, then it is forced to move to the central database again, and this will be applied to all subsystems, the thing that will lead to low performance, slow speed in getting the requested data, long time in obtaining the data and therefore lack of efficiency, which all together make getting the service slow.

Problem formulation

Relational database management systems (RDBMS) are complex server applications that solve the problems of information management. The RDBMS reliably manage large amount of data in a multi-user environment such that users can concurrently access shared data. While it is required to maintain consistent data between users, it is also required to deliver high performance. All these requirements (of managing large amount of data for multi-user environment and concurrently access shared data) need high-quality infrastructure provided by the operating system. Some of the examples that can achieve these targets are: virtual memory management for managing vast amount of physical memory, scalable I/O subsystem, robust / high performance storage subsystem, light-weight inter-process communication, and robust / high performance networking subsystem. Another type of enhancement that we used in order to outperform the system performance is to improve the performance of the database repository model functions (which is the main contribution of this paper), where the repository model is a model that is used in architectural design for the system in software engineering when the data are too huge. It shows the interactions between the subsystems to access the data. So, our main objective is to enhance the first method of the repository model that is called shared database.

The problem of shared database in repository model occurs when a certain subsystem requests for data from the central database. Such problems include: it takes a long time accessing data, like in case of making multiple requests from another subsystems at the same time, and also it is costly and getting data is slow (Philippe, 1998). When large amounts of data are to be shared, the repository model of sharing is mostly using our system to enhance the first method of repository model. When using the repository model mostly, we have to take into consideration that if there is a subsystem that accesses the shared data, then it will take time unless another subsystem requests for the shared database. If multi subsystems request for the central database to be shared, then the following problems will arise: time consumption and less speed, and also starvation or deadlock that is solved by Hofmeister et al. (1999) and Philippe (1998). Accordingly, the questions that this study hopes to answer are:

i. How to reduce the cost of saving the data?
ii. How to increase the performance (speed) of retrieving the data?
iii. How to reduce the time of manipulating and serving the subsystems in a less time?

Major objectives of the proposed solution approach

Modern databases allow for enormous depth and breadth of information to be stored and easily shared where enabling protocols and settings are in place. Email, portable data devices and shared portals further enable information not only to be stored but also accessed and exchanged quickly. This is not to say that achieving good technological arrangements is straightforward or affordable. Getting protocols and settings to a point that

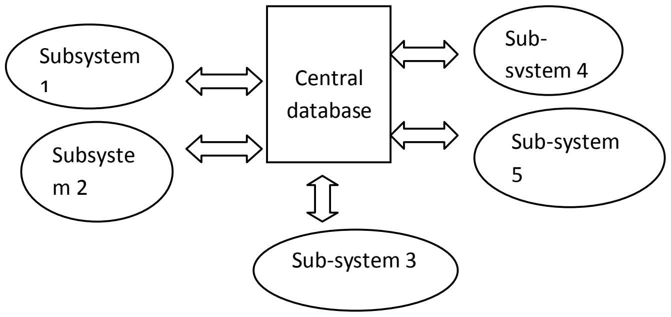

Figure 1. Shared repository model before the suggested enhancement approach.

enables transfer of information between agency systems can require considerable people skills and analysis or purchase of expensive hardware.

Response time and the throughput time are the two scales on which any database server performance can be measured and Microsoft's SQL server is of no difference. Regularly some monitoring tools to measure the performance of Microsoft's SQL sever can be used and it can be measured in seconds for response time and transactions per second for throughput time. Response time is the time which is spent between the initiation and completion of any SQL Query. On the other hand, the throughput time is the number of transactions the server can handle in a given period of time which is usually one second. The performance of SQL server depends upon a number of factors ranging from the hardware to the software. When we talk about the hardware it could be the hard drive on which the database is stored or the server processor. It could also be the physical connection and the network speed; and when we talk about the software it could be the way the application is coded, not only the applications coding but also the SQL queries themselves (http://www.ieee.org).

Our suggested solution that is presented in this paper tries to solve the above mentioned problems in section 2 aiming to achieve the following privileges:

i. Reducing the cost: The buffers are not already created permanently, but we create it according to request, so this will save the cost of the system because there may be this probability that many buffers are created without being useful. Accordingly, this will increase the level of performance by reducing the number of times to access the central database, and it does not make copy of the data that the first subsystems requests for. We store it in the buffer since we save only in the buffer the address reference of a location.

ii. Increasing the performance (speed): In case another subsystem requests for the same data that the first subsystem has requested for, we store it in a buffer, so it can be accessed directly on the buffer with no need to create another buffer.

iii. Reducing the time of manipulating and serving the subsystems: if multi subsystems request for the central database to be shared, we put them in a queue and use an algorithm based on the time the first subsystem requests for it, and then begins the response as they are sorted in that queue. We use the shortest path algorithms which give the optimal solution and less time execution and present the services to all subsystems, thereby avoiding the problems like starvation or deadlock.

Structure of the proposed method

A centralized database system operates by having all participating nodes send their data to a single, centrally-located data repository where it is organized and stored. Instead of going directly to the original source of information (e.g., a county database), users request information from the central repository. The data are brought to a common format that all nodes participating in the sharing initiative need to be able to read or send in. Historically, this was the first architecture model, largely because of limited connectivity and low availability of advanced information systems in local jurisdictions. As these constraints are disappearing, the centralized systems are being replaced or supplemented by other architectures; however, it is too early to completely discount them.

Centralized models are typically associated with a higher overall level of security. By storing data within one large database it is easier to account for data integrity than if the data items were distributed among multiple partners. However, the centralized data pool presents a security challenge in the event that a malicious user does gain access: while the potential for rogues gaining entry into centralized systems is lower than for federated systems, the amount of damage that these users can inflict is potentially higher. Relatively minute changes that are hard-to-track pose a great threat to the integrity of criminal record data. Furthermore, there is higher potential for large scale identity theft and unauthorized data mining.

In Figure1, we illustrate the mechanism of dealing with repository model that is currently used before our development. In this case, the cost is high and the speed of accessing the data is low as a result of the big pressure on the central database, and because search within the central database takes a very long times to obtain the required data from the subsystem. This type of problems exists in the shared database type of the data repository model.

Suggested policy to enhance the model

Searching data records for connections and existing relationships is essential for law enforcement

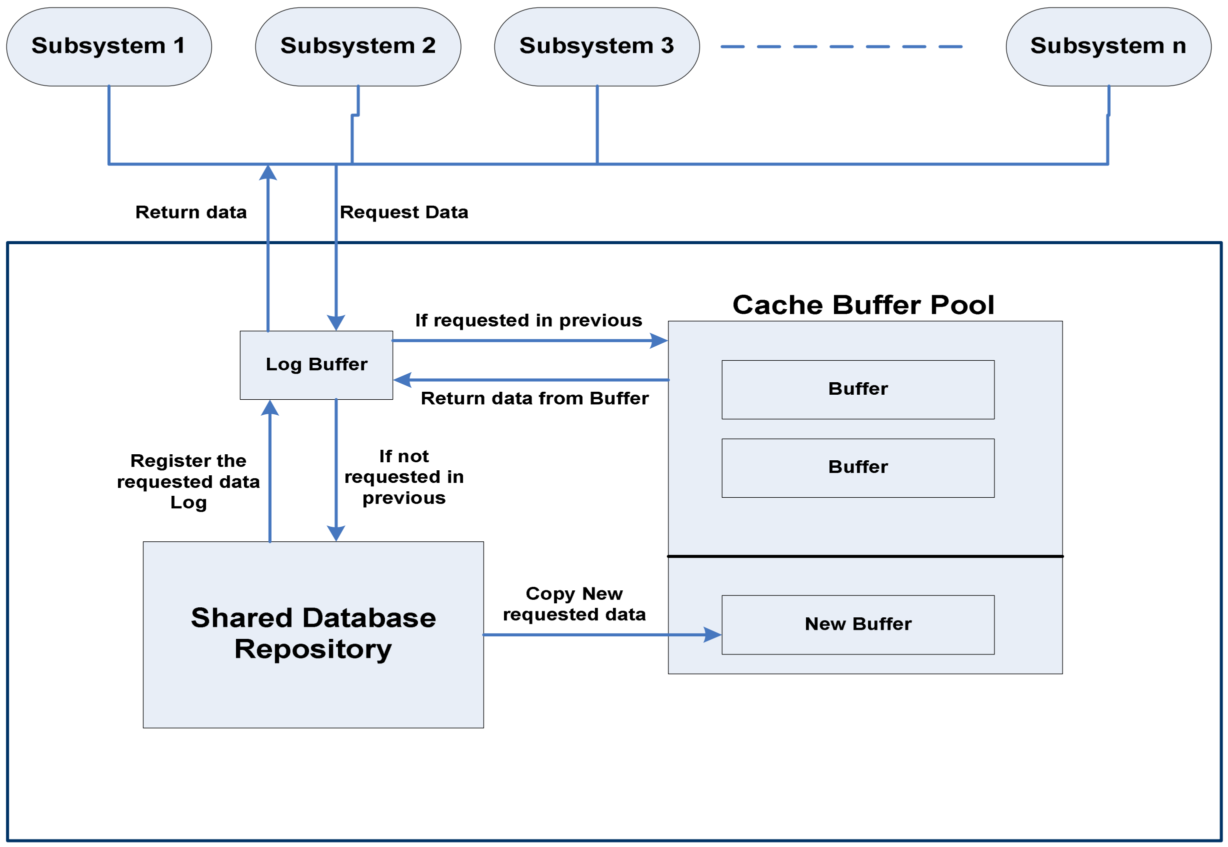

Figure 2. Repository model after the enhancement approach.

investigations. Information overload and confusion are common impediments to efficient analysis, especially when the investigation takes data from different systems and regions (which is essential in preventing drug cartels and terrorist groups). The degree of ease with which the individual user is able to navigate the information sharing network has been a serious concern, for it is the users who ultimately drive (or resist) the system adoption.

In Figure 2, we show the model of shared database after improvement and the mechanism of exchanging data between the sub-systems and the central database, which proves that the speed of accessing the data is high and the time to reach the data is short, making the system overall efficiency high. The specification to build an interoperable digital repository capable of working and sharing information with many databases should cover:-

- Search/Find: The capability to search/find a learning object from the repository Visualization ability can be included.
- Request: To request for a learning object that has been located.
- Retrieve: A located learning object can be retrieved.
- Submit: Sending and storing a learning object to a repository. In other words, adding a learning object to a specific repository.
- Store: Saving a learning object to a specific repository with a global unique identifier in order to assure its localization and manipulation.
- Gather (push/pull): Obtaining metadata
- Information concerning learning objects located in another federated repositories.
- Publish: Providing metadata information about learning objects from a repository to others repositories and systems.

The subsystem requests for data from the central database and searches for data. If it finds the data, it will put it in the buffer then the model will move the data from the buffer to the subsystem number 1. If the subsystem number 2 needs to get data it will go to the data in the buffer and check the data; if it finds it, the model will respond to the subsystem number 2, but if it does not, it will search the central data base. When it finds it, it will put it in buffer 2 after creating it, and then there will be a response to the data in the subsystem number 2.

If the subsystem number 3 requests for the same data, it will search for it in the buffers. If found, the repository model will respond to the subsystem number 3, noting that the size of all the buffers is much less than the size of the central database. There are two cases: the best case is in dealing with data between the sub-systems and the central database. The best case is where all the systems request for the same data stored in one buffer. In this case, the time for data accessing is as short as possible, speed as less as possible, and cost of transferring data as less as possible, leading to improved performance. The worst case takes place when a subsystem requests for data that differ from the data requested by other subsystems; in this case a buffer is established for every subsystem. In this case, it takes longer time to access data and the cost is very high. This leads to reduction in performance. Therefore, the presence of a buffer makes access to the data easier and faster, which leads to better performance.

Implementation of the proposed technique

We designed a simulation tool to achieve the ideas for the proposed approach to achieve the enhancement. So, we will develop our application from two sides: The first one is the server side and the second is from subsystem side where we can use the oracle application server or SQL server 2000/2005 to complete the server side techniques each one installed on MS windows server 2003 and we can use the .net or Oracle forms to complete the client (subsystem) side application (Garlan et al., 1995). In this section, we describe and explain the new tool that will be used to support the main objective of our approach for enhancement to update the repository model where we describe the main structure of this technique and compare between this work, and previous studies in this field. In server side we decided to use the oracle application server because it supports all things that will be used in developing operation of our application from speed in executing any DML (Data Manipulation Language) statement like select, insert, update or delete where the speed of executing these statements is triple of any another database server like SQL server 2000/2005 (from Microsoft corporation).

The proposed algorithm

In algorithm description phase, we will focus on main (database repository) side steps summarized as:

i. Viewing the system global area.
ii. Viewing the buffer size.
iii. Using the shared pool repository.
iv. Using the log buffer.
v. Using the buffer cache pool.

After we have analyzed the storage system, we can make the following decisions to improve the performance:

i. Using automatic sizing of Buffer space.
ii. Adjusting shared pool size for optimal performance.
iii. Increase performance of the log buffer.
iv. Calculating the performance average.
v. Sizing the buffer cache.

The data flow diagram which shows the details about our mechanism is shown in Figure 3. Important data management aspects of scientific dataflow include:

i. Support of complex data structures, such as records containing different attributes of a data object, and sets (collections) of data objects. When combining, merging, and aggregating data, complex compositions of records and sets can arise.
ii. It must be possible to iterate operations over all members of a set.
iii. It must be possible to call external resources and services.
iv. Sub data flows must be supported, that is, one dataflow can be used as a service in another dataflow.
v. Data flows must be specified in a clean, high-level, special purpose programming formalism.
vi. A dataflow can be run several times, often a large number of times, on different inputs.
vii. The data of these different runs must be kept, including input parameters, output data, intermediate results (e.g., from external services), and metadata (e.g., dates).

The last item above is of particular importance and leads to the notion of a dataflow repository: a database system that stores different data flows together with their different runs. Dataflow repositories can serve many important purposes like:-

i. Effective management of all experimental and workflow data that float around in a large laboratory or enterprise setting.
ii. Verification of results, either within the laboratory, by peer reviewers, or by other scientists who try to reproduce the results.
iii. Tracking the provenance (origin) of data values occurring in the result of a dataflow run, which is especially important when external service calls are involved.

Comparison between our Proposed Approach and others

In this section, we will illustrate the difference between proposed study and Philippe’s (1998) study from many different ways like memory usage, response time where we will specify the use of general formulas, and then we use graphs and charts that approximately determine the output ratios. Most of graphs and charts that will be added in this chapter will be drawn by new software application called (Insider 2.1) from (Fourth Elephant company). Notice that the size of data in the all buffers might be equal to the size of data in the central database, and

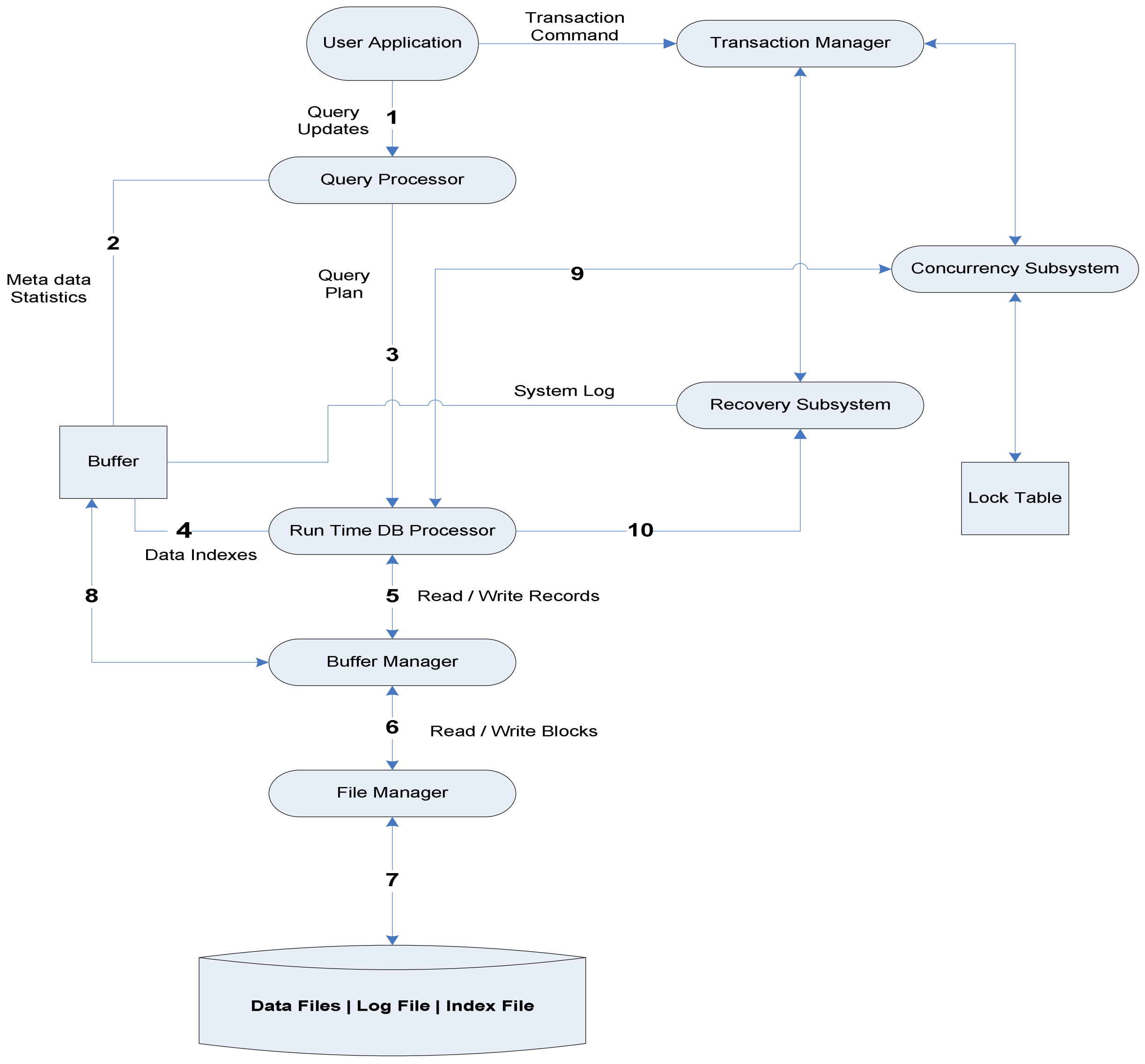

Figure 3. Data flow diagram.

Table 1. Response time related to the proposed simulation tool.

Approach	Average response time in seconds	Maximum response time in seconds	Cumulative response time in seconds
Philippe (1998)	817	5751	58821
Proposed approach	30	275	2131

there are several conditions to be considered while comparing this study with Philippe's (1998) study to measure the performance as a whole:

i. Size of requested data.
ii. Type of requested data (same, different).
iii. Number of requested data.
iv. The starting time to request for data.

The deletion of the data in the buffer in general occurred when the server made restart, so the buffer became clean from data.

Table 1 and Figure 4 show the relation between queries and times about response time in this study such that the number of queries consumes low execution time.

Table 2 shows the summary results of the comparison between proposed study and Philippe's (1998).

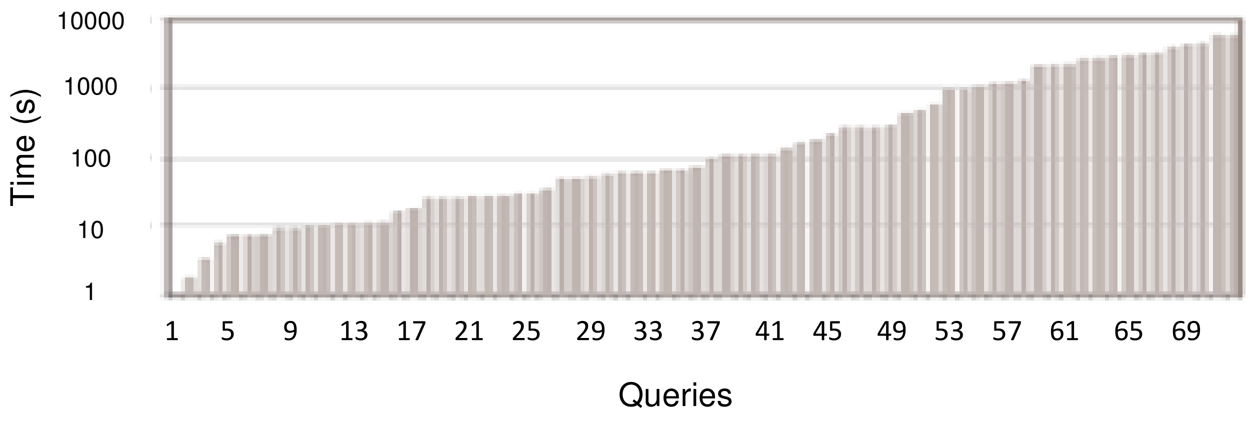

Figure 4. Response time related to Philippe's (1998) study.

Table 2. Comparison between proposed study and Philippe's (1998).

	CPU cost	Disk cost	Database bottleneck	Time cost
Proposed approach	Less	Less	Less	Less
Philippe (1998)	More	More	More	More

Table 3. Workload criteria.

	Low	Medium	High
Users	2	5	13
Transactions	12000	19500	43000
Exec. time	1 h	1.25 h	2 h

The proposed simulation tool or any tool that depends on SGA principles that can be used in Oracle database has been tested with three workload setting criteria: Low, Medium, and High, as described in Table 3. Transactions have been executed by each user independently and simultaneously against a single CPU P-IV 1.8GHz/512 MB of RAM, on Personal edition Oracle 10 g database server. Table shows the transactions between users in many cases by viewing users, transactions and execution time:

Concluded observations

The purpose of this paper is to present a new architectural pattern for the shared repository pattern. This pattern defines a model of communication for software components based on the use of a shared repository. It is a very popular pattern in industrial settings that has been used in numerous and various domains. So, in this paper, the problem of shared database in repository model occurs when a certain subsystem requests for data from the central database. Such problems include: longer time to access data especially in multiple requests from another subsystems at the same time; also high cost and delay in getting data. When large amounts of data are to be shared, the repository model of sharing mostly uses our system to enhance the first method of repository model. If a subsystem accesses the shared data, then it takes time unless another subsystem requests for the shared database to be accessed. We put the data in a buffer which is like a temporary storage that is used to store the data that the first subsystem needs. Then the shared database responds to the request of the other subsystem and begins the response.

In case other subsystem requests for the same data that the first subsystem has requested for, we store them in the buffer, where it can accessed directly without need to create another buffer. In this paper, a new approach to solve the problem of shared database model using a buffer and the proposed model was compared the Philippe's (1998) study. The results showed that the proposed approach makes enhancement by reducing the time, and increasing the speed of access to the data. So, proposed study added an enhancement to shared database pattern. Finally, this paper provides a new approach to access the data efficiently by improving the time, cost and speed to share the data between those subsystems and also to guarantee, that the data will be stored temporarily in the buffer until a new request on data occurs on the buffers.

References

Douglass BP (2005). Software engineering for students,Harlow: Addison-Wesley, 4th edition.

Garlan D, Allen R, Ockerbloom J (1995). Architectural Mismatch or Why it's hard to build systems out of existing parts. Proceedings of 17th International Conference on Software Engineering, Seattle Washington.

Hofmeister C, Robert N, Dilip S (1999). Applied Software Architecture. Addison-Wesley.

Philippe L (1998).Shared repository pattern. Thomson-CSF Corporate Research Laboratory, Domaine de Corbeville, France.

Shaw M, DeLine R, Klein D, Ross T, Young D (1995). Abstractions for Software Architecture and Tools to Support Them. IEEE Transactions on Software Engineering, 21(4).

Shaw M, Paul C (1997). A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. Proceedings of COMPSAC'97, 21st Int'l Computer Software and Applications Conference, Washington, D.C.

Somerville I (2007). Software Engineering. Harlow: Addison-Wesley, 8th edition.

Van der W (1994). CAD Frameworks - Principles and Architecture, Kluwer Academic.

Wakeman L, Jowett J (1993). PCTE - The Standard for Open Repositories, Prentice Hall.

www.eric.ed.gov/ERICWebPortal/recordDetail

www.techgenie.com/.../how-to-improve-microsoft-sql-server-database-performance.

http://www.ieee.org.