Utilization of readily available resources for the design and development of an efficient, cost effective, license free data repository, capable of very large data volume

Chinedu I. Onwuegbuna* and Hyacinth C. Inyiama

Electronic and Computer Engineering Department, Nnamdi Azikiwe University, Awka, Anambra State
*Corresponding author. E-mail: eduextra77@yahoo.com.

Accepted 6th November 2021

The factors taken into consideration in the choice of a suitable database product can be overwhelming. Major relational database products provide about the same level of functionality. Although some products offer more features than others, they all provide the basic relational database management system (RDBMS) functions such as create, read, update and delete. Many small and medium size enterprises require only a large data volume capacity in addition to the basic database functionalities. Changing a database system for an establishment especially when the database is already an established one can be destabilizing. In the quest to avoid provider lock-in scenario, reduce the database management costs, have good control over the database, achieve simplicity of architecture, operation and system administration, many enterprise have resorted to open source database products. This paper is aimed at utilizing readily available resources to design and develop an efficient, cost effective, license free and functional data repository, capable of very large data volume. The developed repository was a (homogenous) virtual repository that is an ensemble of independent identical database files organized into a grid. The needed control and manipulation requirements for the data repository were met by separately developed control algorithms. These algorithms make up the virtualized master data management layer. This data management layer feeds and manages the requests to the repository, taking into consideration the peculiar structuring and organization of the repository. This developed repository can provide medium and small establishments a low cost yet functional alternative to the costly larger RDBMS.

Key words: Repository, database, algorithms, NoSQL, schema, RDBMS, big-data, scalability.

INTRODUCTION

A database according to Peter and Coronel (2002) is a shared, integrated computer structure that houses a collection of end user raw facts/data and metadata through which the data are integrated. Philip and Joseph (2005) defined a database as a structure that can store information about multiple types of entities, the entities' attributes and the entities' relationships. Raghu and Johannes (2003) described a database as a collection of data, typically describing the activities of one or more related organizations. Fred and Jeffrey (1985) on the other hand defined it as a shared collection of interrelated data designed to meet the varied information needs of an organization. Peter and Coronel (2002) went further to describe a database as a well organized electronic filing cabinet where a powerful software known as Database Management System (DBMS) helps to manage the cabinets contents. Arshi (2017) and Azhar and Meiryani (2019) all agreed that a DBMS is a collection of software programs that manages the structure of the database, that it is the link between the physical data files and the application programs that manage access to the stored data and make it possible to share the database content among multiple application and users. Peter and Coronel (2002) further stated that the functions of a database include data dictionary management, data storage management, data transformation and presentation, security management, multi user control, backup and recovery management, database access language, application programming interfaces and database communication interfaces. In the quest to get an enhanced data management, several database models/systems have been developed. A database model is a collection of logical constructs used to represent the data structure and data relationships found in the database. Fred and Jeffrey (1985), Rai and Pramod (2015) and Peter and Coronel (2002), all identified the following models: the Hierarchical database model, the Network database model, the Relational database model and the Object oriented database model. According to Ward and Dafoulas (2006), the relational database systems RDBMS are by far the most common type of database system around today. This is due to the wide acceptance of this model for traditional business applications. Again, a well designed relational database can provide appropriate data storage and retrieval facilities that will serve over a long time. However, relational database model has difficulty in handling huge-data, the type usually found in modern applications such as web, mobile and gaming. According to Michael et al. (2015), performance requirements, schema flexibility demands, scalability requirements due to increase in web oriented applications has presented significant challenges for the traditional relational databases. Moniruzzaman and Syed (2013), Michael et al. (2015), and Biswajeet et al. (2014) all pointed out that the need for large data volume storage, the need for near real time transaction response in transaction processing, the need for data schema flexibility and for applications to scale horizontally with predictable licensing and hardware cost has resulted in the introduction of more fitting database models grouped under the name NoSQL (Not Only SQL) database. Among these are the Key-value database model, the Document, Graph, In-memory and Search database models. These database models are non-relational and are optimized for the kind of job it is to handle. Michael et al. (2015) stated that the old/previous database models which include the Hierarchical database model, the Network database model, the Relational database model and the Object Oriented database model were not designed to take advantage of the inexpensive storage and the extra processing power available these days neither are they designed to cope with flexibility and scalability challenges posed by modern applications. Hira and Roshan (2020) and Moko and Asagba (2020) pointed out that these NOSQL technologies seek to solve and address these challenges and these technologies do not use the database table as the data storage structure and they have an efficient schema for unstructured data.

There exist today, many relational database products that differ on performance, ease of database administration, functionality and price. Also, there are many RDBMS suppliers like Oracle, Microsoft, IBM, Hitachi, Hewlett-Packard, Fugitsu, and Sybase NCR Tetradata to mention but a few. The selection of a suitable database product can be a very difficult process because so many factors must be taken into consideration such as functionality, data model, usability, visualization and reporting, security, support and development, ease of integration with other software systems, scalability, cost, suitability, mode of access, efficiency, implementation and service costs, adaptability, predictability, etc. Other factors in making a choice of database product are the database products' conformity with the users' data requirements which invariably depends on the problem the database management system is intended to solve. Other factors include performance, total cost of ownership, simplicity of architecture, openness of source code, quality of support by vendor, maturity of database and its vendor, simplicity of the operation and system administration and above all, the ability to know what to do to recover when the database system goes down and there is no one else to get assistance from.

Today, to stay away from provider lock-in scenario and to reduce the database management costs, many enterprises have resorted to open source database products. According to Rod Stephens (2007), the entire major relational database products provide about the same level of functionality. Some products provide more features than the others but all the products provide the basic RDBMS functions such as to create, read, update and delete data, collectively known as CRUD. These functions form the foundation of a well-organized system that promotes consistent treatment of data. It can be overwhelmingly destabilizing for an establishment to change a database system especially when the database is already an established one for the establishment. Many a times the reason for database platform switch are centered on factors like application compatibility as desired by the user, the need for more capacity especially with respect to data volume, licensing issues, total cost of ownership, mergers and acquisition, etc. Conformity with the users' data requirements greatly depends on the problem the RDBMS is intended to solve. Architectural simplicity, openness of source code, simplicity of the database operation and administration, all these attributes will enable the user understand and know what to do to recover in case the database system crashes.

Most small and medium size enterprises need only a large data volume capacity in addition to the basic database functionality. Taking all these into consideration, this work is aimed at developing an efficient, cost effective, license free data repository that is capable of very large data volume using readily available materials. The term (Data Repository) as used in this contest refers to a location where data is stored and maintained. The repository developed here is a (homogenous) virtual repository that is an ensemble of independent identical database files organized in a grid like fashion but will operate as a single repository. The user or application that will accesses and use this repository will have the feel as that when accessing a single database.

The choice of material for use as the building block for the proposed data repository that will suit this design was considered on the basis of the following factors: Reliability, Adaptability, Scalability, Predictability and Manageability.

Reliability
This attribute refers to the consistency in performance of the product according to its specifications and how easy it is to secure and safeguard data in case of trouble so that critical applications can proceed uninterrupted.

Adaptability
This attribute refers to how easily bendable is this product of choice to the exact blend of responsibility of the users' requirements. This includes the ability to adapt to rapidly changing assignments, without imposing costly hardware upgrades or new purchases.

Scalability
The product of choice must to a reasonable extent able to support a growing number of users, data volume and transactions without demanding new hardware costs and still deliver an acceptable response time.

Predictability
This product of choice should be capable of a predictable service level in supporting varying workloads. It should easily adjust when confronted with expanding demands with respect to transactional load, number of users and data volume.

Manageability
The product set up must be fast devoid of complicated installation procedure, specialized skill or extensive training. The long term total ownership cost which includes software license, administrative costs, running costs, staffing expenditures, etc., must be kept at a minimum.

METHODOLOGY
In the light of the aforementioned, the Microsoft Access RDBMS was chosen. The Microsoft Access RDBMS will not be used in the conventional ways as an Access database engine (that is, to use the Access intrinsic/native database design objects and functionalities) but rather it is to be used in an unconventional way as the data repository's building blocks. The Microsoft Access RDBMS is easy to use and also easy to propagate. This characteristic nature also makes it amenable to this repository design. The necessary control and data manipulation requirements for the data repository were met by

separately developed control software algorithms and routines that act as a virtualized master data management layer. This data management layer manages the requests to the repository, taking into consideration the peculiar structuring and organization of the repository. The data repository so organized compensated effectively for the data capacity limitation of the Access RDBMS. The data capacity of the data repository in effect was limited by the hardware used to set it up. For example, a 120 gigabytes hard disk drive can comfortably house and manage a data repository with over 100 gigabytes of data. Again, the data management layer also provided the other features like log-keeping of all updates to the repository which expensive RDBMS like Oracle, etc., provide. The data management algorithm could easily take advantage of the various Microsoft Access RDBMS functionalities when needed such as in transaction management, etc. The data recovery in case of failure for the repository was achieved by periodic data-backups, alongside log-keeping which was scheduled at intervals to limit data loss. The classical username and password security arrangement was used. The Microsoft Visual Basic Programming Language was used to develop the data management controlling routines. This language of choice should support a universal data access capability that is flexible enough to be adapted to suite the data repository. Microsoft visual basic programming language choice enabled one to take advantage of the Microsoft's' strategy for universal data access, which is delivered through a common set of modern, object oriented interfaces based on Microsoft's component object model (COM) for access to data both for relational and non-relational data sources.

Data repository design

The data repository developed is a (homogenous) virtual repository that is an ensemble of independent identical database files that is referred to in this work as buckets. These buckets were addressed and coordinated by routines on the application layer. The buckets are named such that their individual names facilitated the addressing and referencing of each bucket. Each bucket (database file) is assumed to be capable of providing the basic functionalities of an RDBMS such as data storage, retrieval, update, data directory/dictionary, transactional integrity, data recovery services, concurrency control, security mechanisms, data communications interface, and data integrity services. Each bucket may or may not have a large data storage capacity.

The principle/approach adopted for naming the buckets is as follows. Each bucket's name is made up of three parts namely: the PREFIX, GENE and DIFFERENTIATOR.

The PREFIX is a sequence of characters placed at the beginning of this coded bucket's name. The combination of the PREFIX and the GENE are the part of bucket's name that enables the algorithms and control routines in the virtualized master data management layer to identify a homogenous group in the repository. The PREFIX is static throughout a homogenous group.

The GENE is the (ending) part of the bucket's name. The GENE is also static throughout a homogenous group.

The DIFFERENTIATOR is the (middle) part of the bucket's name that will enable the control routines of the virtualized master data management layer to identify and differentiate among the members of a homogenous group in the data repository. The DIFFERENTIATOR varies and its value increases with every new additional bucket in a homogenous group. Each individual bucket's name is then the concatenation of the PREFIX, DIFFERENTIATOR and the GENE, respectively. For example, if the PREFIX is (friget), the GENE (12345) and the DIFFERENTIATOR is 76 then the bucket's name will be friget7612345.

For proper management and coordination of this data repository, there is the need to also design a control bucket that will hold the data repository storage units' control information. These control information was made available to be globally referenced by all the application program routines that will need to access the data repository to read, write and update records, etc.

The data repository storage unit control information that was held in the control bucket includes that for every GENE (that is, a homogenous group), the following are documented:

(a) The Number of Existing Storage Units (number of existing buckets)
(b) The Active Storage Unit (the active bucket)

It is only the control information about the buckets needed by the control algorithms that are stored in the control DB. This enabled the various control algorithms to access them to perform their functions properly.

The following algorithms were used to manage and coordinate the repository operations. These are (Tables 1 and 2):

(1) The algorithm for read routine
(2) The algorithm for write routine (ADDNEW) and modification processes

Table 1. The algorithm for data READ routine.

S/N	Algorithm for Read routine	Algorithm descriptions
1	• Dimension Variable for documenting the caller routine / form • Dimension Variable for documenting Number of Storage Units with the GENE under consideration • Dimension Variable for documenting the Current Active Storage Unit for Storage Units with the GENE under consideration • Dimension Tagging Variable for Tracking and synchronizing a particular patient's record	Dimension needed variables
2	• Synchronize particular patient's records using the patents Tagging Variables	
3	• Dimension Integer Variable for loop control • Establish connection to Control bucket and target the Repository Storage Unit Control Information (table) • Do Until End Of File - Populate Variable for No of Existing Storage Units - Populate Variable for Current Active Storage Units Loop • Close Connection	Establish connection to Control bucket and target the gene of the desired homogenous group in the Data Repository Storage Unit Control Information (Table) Get control information on number of existing storage units Get control information on the active storage unit Close Connection
4	• Prepare Temporary cache/ Microsoft Hierarchical FlexGrid	
5	• Dimension Loop control Integer Variable • For VAR1 =1 to (Total Number of existing Storage Unit for the GENE under consideration) control information ➢ Establish connection to Bucket (Storage Unit VAR1) and target the Number of existing Records (control information) ○ Do Until End Of File - Get control information on number of existing records ○ Loop ➢ Close Connection • Next VAR1	Dimension Integer Variable for loop control Set up a FOR TO NEXT LOOP to run from the first member of a homogenous group (GENE) housing the (desired records verification information) to the last member of the group Establish connection to Buckets and target the Number of existing desired Records (proper documentation check information) Get number of existing records (proper documentation check information) Close Connection End FOR TO NEXT LOOP *(end of retrieving of proper documentation check information)*

Table 1. Continue

6	• Populate Visual Assistant display segment	Populate Visual Assistant display segment(if used)
7	• For VAR2 =1 to (Total Number of existing Storage Units for GENE under consideration) ➢ Establish connection to Bucket (Storage Unit VAR2) and target the actual Patients Records ○ Do Until End Of File - Get the needed Patients Records - Populate Temporary cache/ Microsoft Hierarchical FlexGrid ○ Loop ➢ Close Connection • Next VAR2	Set up a FOR TO NEXT LOOP to run from the first member of the homogenous group (GENE) housing the (desired records) to the last member of the group Establish connection to the buckets targeting the desired record *(using string processing to tag the loop variable to the buckets differentiator)* Read all data from all the existing buckets, the desired records via the loop setup Close Connection End FOR TO NEXT LOOP *(end of reading/retrieving of records)*
8	• Check for record documentation Correctness and report/act accordingly	Check for record documentation Correctness and report/act accordingly Populate Visual Assistant display segment(if used)

Table 2. The algorithm for data WRITE routine (ADDNEW) and Modification processes.

	Algorithm for write routine	Algorithm descriptions
1	• Dimension Variable for documenting the caller routine / form • Dimension Variable for documenting Number of Storage Units with the GENE under consideration • Dimension Variable for documenting the Current Active Storage Unit for Storage Units with the GENE under consideration • Dimension Tagging Variable for Tracking and synchronizing a particular patient's record • Dimension Log/Failsafe Variables • Dimension Grid/Temporary cache Variables	Dimension needed variables
2	• Synchronize particular patient's records using the patents Tagging Variables	
3	*"Create This Add New/ Update/ Write Log Routine as a subroutine to be referenced later in the write routine algorithm"* • Dimension Integer Variable for Current Active Log Storage Unit • Populate the Variable for Current Active Log Storage Unit • Establish connection to Log Storage Unit/ Bucket and target the Current Active Log Storage Unit *(AddNew connection)* ➢ With RecordSet - Populate Variables for Log Storage Unit Update *(New Entries)* ➢ End With	*Create This AddNew/ Update/ Write Log Routine as a subroutine to be referenced later by the data repository write routine algorithm* Dimension Integer Variable for Current Active Log Storage Unit Populate the Variable for Current Active Log Storage Unit Establish connection to Log Storage Unit/ Bucket and target the Current Active Log Storage Unit *(AddNew connection)* Populate Log Storage Unit (New Log Entries) Close Connection

Table 2. Continue

4	• Dimension Integer Variable for loop control • Establish connection to Control bucket and target the Repository Storage Unit Control Information (table) ➢ Do Until End Of File - Populate Variable for No of Existing Storage Units - Populate Variable for Current Active Storage Units - Populate Variable for No of Existing Fail Safe Units - Populate Variable for Current Active Fail Safe Unit ➢ Loop • Close Connection	Dimension Integer Variable for loop control Establish connection to Control bucket and target the gene of the desired homogenous group in the Data Repository Storage Unit Control Information (Table) Get control information on number of existing storage units (for desired record) Get control information the active storage unit (for desired record) Get control information on number of Log/Fail Safe existing storage units Get control information the active Log/Fail Safe storage unit Close connection
5	• Prepare Temporary cache/ Microsoft Hierarchical Flex Grid	Prepare Temporary cache/ Microsoft Hierarchical Flex Grid(if desired)
6	• Dimension Loop control Integer Variable • For VAR1 =1 to (Total Number of existing Storage Unit for the GENE under consideration) control information ➢ Establish connection to Bucket (Storage Unit VAR1) and target the Number of existing Records (control information) ○ Do Until End Of File - Get control information on number of existing records ○ Loop ➢ Close Connection • Next VAR1 • Populate Visual Assistant display segment	Dimension Loop control Integer Variable Set up a FOR TO NEXT LOOP to run from the first member of a homogenous group (GENE) housing the (desired records verification information) to the last member of the group Establish connection to Buckets and target the Number of existing desired Records (proper documentation check information) Get number of existing records (proper documentation check information) Close Connection End FOR TO NEXT LOOP *(end of retrieving of proper documentation check information)* Populate Visual Assistant display segment(if used)
7	• For VAR2 =1 to (Total Number of existing Storage Units for GENE under consideration) ➢ Establish connection to Bucket (Storage Unit VAR2) and target the actual Patients Records ○ Do Until End Of File ○ Get the needed Patients Records ○ Populate Temporary cache/ Microsoft Hierarchical Flex Grid ○ Loop ➢ Close Connection • Next VAR2	Set up a FOR TO NEXT LOOP to run from the first member of the homogenous group (GENE) housing the (desired records) to the last member of the group Establish connection to the buckets targeting the desired record *(using string processing to tag the loop variable to the buckets differentiator)* Read all data from all the existing buckets, the desired records via the loop setup Close Connection End FOR TO NEXT LOOP *(end of reading/retrieving of records)*

Table 2. Continue

8	• Check for record documentation Correctness and report/act accordingly	Check for record documentation Correctness and report/act accordingly
9	• Validate if entries were made correctly • Initialize Log variables *(for first stage of updates)* • Request Clearance for Saving/ Update of Records *(Add New)* • For VAR3 =1 to (Total Number of existing Storage Units for GENE under consideration) ➢ Establish connection to Bucket (Storage Unit VAR3) and target the actual Patients Records control Information *(Add New/ Write)* ○ Do Until End Of File ○ Get the needed Patients Records control Information Records ○ Update ○ Loop ➢ Close Connection • Next VAR3	Initialize log variables for First Stage Updates *(i.e. the desired records verification information)* Request Clearance for Saving/ Update of Records *(Add New)* Set up a FOR TO NEXT LOOP to run from the first member of the homogenous group (GENE) housing the (desired records verification information) to the last member of the group Establish connection to the buckets targeting the desired record's verification Information *(using string processing to tag the loop variable to the buckets differentiator)* Update the desired Record verification Information Close connection End FOR TO NEXT LOOP*(end of first stage updates*
10	• Reinitialize Log variables (*for second stage of updates)* • For VAR4 =1 to (Total Number of existing Storage Units for GENE under consideration) ➢ Establish connection to Bucket (Storage Unit VAR4) and target the actual Patients Records *(Add New/ Write)* ○ Do Until End Of File - Get the needed Patients Records - Update *(Add New)* ○ Loop ➢ Close Connection • Next VAR4 • Update Temporary cache/ Microsoft Hierarchical Flex Grid and UI • Report Task Completion	Reinitialize Log variables for second stage of updates *(i.e. the actual desired records)* Set up a FOR TO NEXT LOOP to run from the first member of a homogenous group housing the (desired records verification information) to the last member of the group Establish connection to the buckets targeting the actual desired record this time *(using string processing to tag the loop variable to the buckets differentiator)* Update the desired Records Close connection End FOR TO NEXT LOOP*(end of second stage updates)* Update Temporary cache/ Microsoft Hierarchical Flex Grid and User Interface *(if used)* Report Task Completion

RESULTS AND DISCUSSION

A functional data repository was developed as an ensemble of independent identical access database files. The building blocks of this repository were named such that the data management layer can properly identify and manipulate them. The developed repository is cost effective, license free, and capable of very large data volume. The needed control and manipulation requirements for the data repository were met by separately developed control algorithms. These algorithms make up the virtualized master data management layer. These algorithms/data management layer feeds and manages the requests to the repository/database pool, taking into consideration the peculiar structuring and organization of the data repository. This developed algorithm was applied in developing a Data Repository for a Hospital Information System (HIS). The Shell (SPDC) Information and Communication Technology (ICT) Research Center, Delta State University, Abraka, located in Delta State assisted in the HIS software deployment, testing and algorithm performance evaluation. The HIS was developed as a Server-Client application with the client side developed as a rich/thick client. The ICT center provided a suitably networked environment, a local area network (LAN) that is capable of data speed of up to 100 Mbps for the computers connected using category 5 network cables and 54 Mbps for the computers connected via the wireless access. HIS client software was installed in 20 desktop computers that were connected using the category 5 network cables and 5 laptop computers that were connected using the wireless access, the repository was installed in a server computer and monitored for repository algorithm effectiveness. Four doctors, three laboratory scientists, five nurses, three administrative staff, fifteen persons were involved in the testing to simulate a typical hospital session. It was observed that the developed algorithm was effective giving an observed response time in the range of 0.1 to 2.5 s which is within acceptable response time range for such an application. This data repository organization effectively eliminated the data capacity limitation of two gigabytes of the conventional Access RDBMS. The data capacity of the data repository in effect was limited by the hardware capacity used in setting it up. The capacity of the repository can be increased by the addition of more access database and updating the control information in the control DB. This approach to database design and use can provide small and medium sized establishments an alternative to the costly larger RDBMS with a low price tag, especially in scenarios where the major requirement of the establishment is a large data capacity with the basic database functionality.

Conclusion

This work demonstrated how a functional data repository that is capable of large data volume can be effectively developed using free inexpensive RDBMS like access database. It also demonstrated a reengineering approach to database use.

REFERENCES

BM Moniruzzaman and Syed Akhter Hossain, (2013). NoSQL Database: New Era of Database For Big Data Analytics- Classification, Characteristics and Comparison, International Journal of Database Theory and Application, Vol 6(4)

Arshi Gouhar, (2017). Database Management System, International Journal of Engineering Science and Computing, Vol 7(5)

Azhar Susanto and Meiryani, (2019). Database Management System, International Journal of Scientific Technology Research, Vol 8(6)

Biswajeet Sethi, Samaresh Mishra and Prasant ku. Patnaik, (2014). A Study of NoSQL Database, International Journal of Engineering Research and Technology, Vol 3(4)

Fred R McFadden and Jeffrey A. Hoffer, (1985). Database Management, United States: The Benjamin/Cummins Publishing Company, Pp 66-68

Hira Lai Bhandari and Chitrakar (2020). Comparison of Data Migration Techniques from SQL to NoSQL Database, Journal of Computer Engineering and Information Technology, Vol 9(6)

Michael Madison, Mark Barnhill, Cassie Napier, Joy Godin, (2015). NoSQL Database

Technologies, Journal of International Technology and Information Management. Vol 24(1)

Moko Anasuodei and Asagba Prince Oghenekaro, (2020). Big Data and NoSQL Databases Architecture: A Review, International Journal of Applied Sciences and Mathematical Theory, Vol 7(1)

Patricia Ward and George Dafoulas, (2006). Database Management Systems, Thomson Learning

Peter Rob and Carlos Coronel, (2002). Database Systems Design, Implementation, and Management, Fifth Edition, USA: Course Technology, Thomson Learning, Pp 611-649.

Philip J. Pratt and Joseph J Adamski (2005). Concepts of Database Management, Fifth Edition, Thompson Course technology

P.K.Rai and Pramod Singh, (2015). Studies and Analysis of Popular Database Models, International Journal of Computer Science and Mobile Computing, Vol 4(5)

Raghu Ramakrishaan and Johannes Gehrke (2003). Database Management Systems, Third Edition, New York City: McGraw-Hill, p4-24

Rod Stephens, (2007). Expert One-On-One Visual Basic 2005 Design and Development, New Jersey: Wiley Publishing, Inc, Pp 133-145