International Journal of Scientific Engineering and Research (IJSER)
www.ijser.in
ISSN (Online): 2347-3878
Volume 2 Issue 2, February 2014

Database System Concepts, Implementations and Organizations-A Detailed Survey

Udoka Felista Eze[1], Chukwuemeka Etus[2], Joy Ebere Uzukwu[3]

[1]Federal University of Technology Owerri, School of Management Technology,
P.M.B 1526 Owerri, Imo State, Nigeria

[2]Federal University of Technology Owerri, School of Management Technology,
P.M.B 1526 Owerri, Imo State, Nigeria

[3]Federal Polytechnic Oko, Faculty of Applied Science, Department of Computer Science,
P.M.B. 21 Aguata Local Government Area, Anambra State

Abstract: ***The inclusion of database and database management system in the use of data is of paramount importance. Database design, access and management are equivalent to plumbing. Like plumbing, there are dozens of segments that must be put together before the whole thing works. In fact, database systems are now so common and form such an integral part of our day-to-day life that often we are not aware we are using one. The methodology of this survey covers both industrial and academic environments, and divided into three phases: (i) a conceptual database design phase, in which a model of the data is developed and used in an organization independent of all physical considerations; (ii) a logical database design phase, in which the developed relational model of the data independent of any particular DBMS and other physical considerations is developed; (ii) a physical database design phase, in which the implementation in the target DBMS, such as Microsoft Access, Microsoft SQL Server, Oracle, DB2, or Informix is realized. Also, examined, are what constitutes a database system and how this tool can bring great benefits to any organization that chooses to use one; with a deep look at all the important aspects of database systems design, implementation and management including conceptual database, Schemas, Database Management Systems, Implementation Database, Logical and Physical Database Organizations and structure, with their basics, problems or shortcommings, benefits, requirements, components, types and roles, implementations and organizations.***

Keywords: Database, Architecture, Interface, Schema.

1. Introduction

The backbone of an information management system (IMS) is the database. It has fundamentally changed the way many work. The developments in information technology over the last few years have produced database systems that are more powerful and more intuitive to use, and some users are creating databases and applications without the necessary knowledge to produce an effective and efficient system.

It has been estimated that one third of a data model consists of common constructs that are applicable to most users and the remaining two thirds are user-specific (customized). Thus, most database design work consists of recreating constructs that have already been produced many times before in other companies. The models featured may not represent a company exactly, but they may provide a starting point from which a more suitable data model can be developed that matches a company's specific requirements. Some of the models service common business areas including Customer Order Entry, Inventory Control, Project Management, Human Resource Management, and Payroll Management, amongst others. Increasingly, users are standardizing the way in which they model data by selecting a particular approach to data modeling and using it throughout their database development projects.

1.1 Database Overview

A database is any logically coherent collection of data organized for storage and retrieval by computers, as a single, possibly large, repository of data that can be used simultaneously by multi-users [1]. Databases provide a high level of structure for the collection of data. Structured collection of data leads to a consistency that is essential for the operation of an IMS. Data within a database are captured electronically with a relatively small amount of storage space compared with conventional paper records. Databases also allow the use of queries and search tools, which make the retrieval of information from even large amounts of data fast and highly specific.

The two main types of databases are flat-file and relational databases, in both of which the data are placed in tables with columns and rows. However, in a flat-file database, the data are placed within a single, spreadsheet-like table. In a relational database, each of several tables contains a specific type of information, and the tables are linked by primary and secondary keys. Flat-file databases can be very efficient when there are simple one-to-one relationships between the data (eg, one patient to one medical record number [MRN]) [2]. When the data have one-to-many or many-to-one relationships (e.g., one physician to many patients or vice versa), it is best to use a relational database. Relational databases are preferred to flat-file databases for all but simple applications that are limited in scope.

A popular high-level data model used in logical database design is based on the concepts of the Entity–Relationship (ER) model. Currently there is no standard notation for an ER model covering database design for relational DBMSs

tending to use one of two conventional notations such as: (i) Chen's notation, consisting of rectangles representing entities and diamonds representing relationships, with lines linking the rectangles and diamonds; (ii) Crow's Feet notation, again consisting of rectangles representing entities and lines between entities representing relationships at the end of a line represents a one-to-many relationship [3].

The database is now such an integral part of our day-to-day life that often we are not aware we are using one. A database application is a computer program that interacts with the database in some way, while database system is a collection of database applications that interacts with the database along with the DBMS and the database itself. Hence, the basic functions of a database are provided by a DBMS through a database engine; a software system that manages how data is stored and retrieved [4]. For example, Microsoft Jet is a database engine which is a subsystem of Visual Basic and Microsoft Access for stand-alone implementation; while Microsoft SQL Server is for Client - Server implementation.

Users interact with the database through a number of database applications that are used to create and maintain the database and to generate information. These programs can be conventional batch applications or, more typically nowadays, online applications. The database applications may be written in a third-generation programming language such as C++ or Java, or in some higher-level fourth-generation language. The physical structure and storage of the data are managed by the DBMS [5].

2. Conceptual Database

The conceptual database comprises some theoretical frameworks of database developments, architectures and models. These are examined below.

2.1 Three-level ANSI-SPARC Architecture

Date [6] opines that an early proposal for a standard terminology and general architecture for database systems was produced in 1971 by the Data Base Task Group (DBTG) appointed by the Conference on Data Systems and Languages (CODASYL). The DBTG recognized the need for a two-level approach with a system view called the schema and user views called sub-schemas. The American National Standards Institute (ANSI), and Standards Planning and Requirements Committee (SPARC) produced a similar terminology and architecture in 1975. ANSI-SPARC recognized the need for a three-level approach with a system catalog. Although the ANSI-SPARC model did not become a standard, it still provides a basis for understanding some of the functionality of a DBMS.

The ANSI-SPARC model identified three distinct levels at which data items can be described: an external level, a conceptual level, and an internal level, as depicted in Figure 1. The way users perceive the data is called the external level. The way the DBMS and the operating system perceive the data is the internal level, where the data is actually stored. The conceptual level provides both the mapping and the desired independence between the external and internal levels. The objective of this 'three-level architecture' is to separate each user's view of the database from the way it is physically represented [5].

There are several reasons why the separation is desirable: (i) Each user should be able to access the same data, but have a different customized view of the data. (ii) Users should not have to deal directly with physical database storage details, such as file structures or indexing. (iii) The Database Administrator (DBA) should be able to change the database storage structures without affecting the users' views. (iv) The internal structure of the database should be unaffected by changes to the physical aspects of storage, such as moving to a new storage device. (v) The DBA should be able to change the conceptual structure of the database without affecting all users.

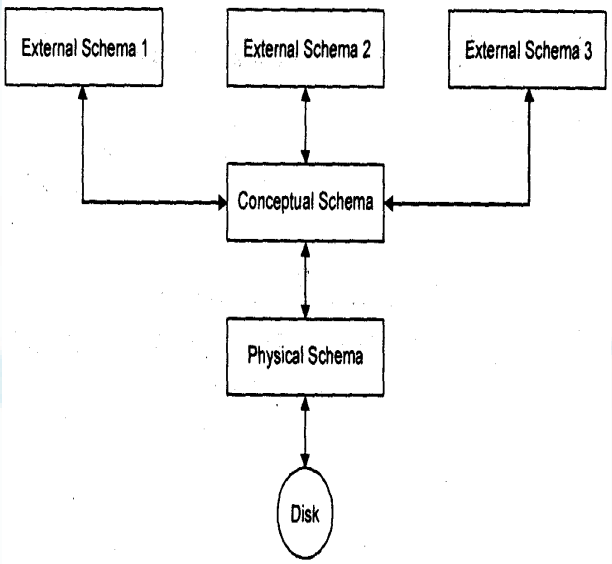

Figure 1: The ANSI–SPARC three-level architecture

2.1.1 External Level

The external level consists of a number of different external views of the database. Each user has a view of the 'real world' represented in a form that is familiar for that user. The external view includes only those entities, attributes, and relationships that the user is interested in. Other entities, attributes, or relationships that are not of interest may be represented in the database, but the user will be unaware of them. In addition, different views may have different representations of the same data. For example, one user may view dates in the form (day, month, year), while another may view dates as (year, month, day). Some views might include derived or calculated data, data not actually stored in the database as such but created when needed [7].

2.1.2 Conceptual Level

The middle level in the three-level architecture is the conceptual level. This level contains the logical structure of the entire database as seen by the DBA. It is a complete view of the data requirements of the organization that is

International Journal of Scientific Engineering and Research (IJSER)
www.ijser.in
ISSN (Online): 2347-3878
Volume 2 Issue 2, February 2014

independent of any storage considerations. The conceptual level represents:

(i) All entities, their attributes, and their relationships;
(ii) The constraints on the data;
(iii) Semantic information about the data;
(iv) Security information.

The conceptual level supports each external view, in that any data available to a user must be contained in, or derivable from, the conceptual level. However, this level must not contain any storage-dependent details. For instance, the description of an entity should contain only data types of attributes (for example, integer, float, character) and their length (such as the maximum number of digits or characters), but not any storage considerations, such as the number of bytes occupied [8].

2.1.3 Internal Level

The internal level covers the physical implementation of the database to achieve optimal runtime performance and storage space utilization. It covers the data structures and file organizations used to store data on storage devices. It interfaces with the operating system access methods (file management techniques for storing and retrieving data records) to place the data on the storage devices, build the indexes, retrieve the data, and so on. Below the internal level there is a physical level that may be managed by the operating system under the direction of the DBMS. Some DBMSs take advantage of many of the operating system access methods, while others use only the most basic ones and create their own file organizations. The physical level below the DBMS consists of items only the operating system knows, such as exactly how the sequencing is implemented and whether the fields of internal records are stored as contiguous bytes on the disk or not [9].

2.1.4 Schemas and Instances

The overall description of the database is called the database schema. There are three different types of schema in the database and these are defined according to the levels of abstraction of the three-level architecture, as illustrated in Figure 2. While there is an external schema for each user view of the database, there is only one conceptual schema and one internal schema per database. It is important to distinguish between the description of the database and the database itself. The description of the database is the database schema. The schema is specified during the database design process and is not expected to change frequently. However, the actual data in the database may change frequently; for example, it changes every time we insert details of a new member of staff or a new product detail. The data in the database at any particular point in time is called a database instance. Therefore, many database instances can correspond to the same database schema.

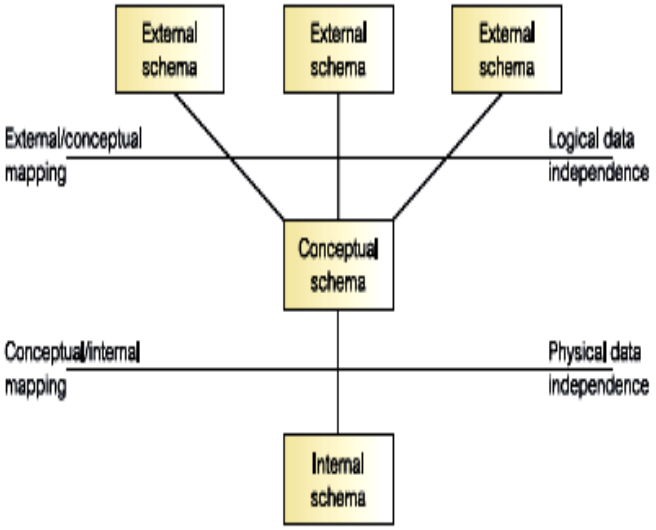

Figure 2: Schemas & data independence in ANSI-SPARC three-level architecture

2.1.5 Data Independence

A major objective for the three-level architecture is to provide data independence, which means that upper levels are unaffected by changes to lower levels. There are two kinds: logical data independence and physical data independence. Changes to the conceptual schema, such as the addition or removal of new entities, attributes, or relationships, should be possible without having to change existing external schema or having to rewrite database applications. Clearly, the users for whom the changes have been made need to be aware of them, but what is important is that other users should not be. Changes to the internal schema, such as using different file organizations or modifying indexes, should be possible without having to change the conceptual or external schemas. From the users' point of view, the only effect that may be noticed is a change in performance. In fact, deterioration in performance is the most common reason for internal schema changes. Figure 2 illustrates where each type of data independence occurs in relation to the three-level architecture.

2.1.6 Database Security [10]

A secure database must satisfy the following requirements (subject to the specific priorities of the intended Application):

i. It must have physical integrity (protection from data loss caused by power failures or natural disaster),
ii. It must have logical integrity (protection of the logical structure of the database),
iii. It must be available when needed,
iv. The system must have an audit system,
v. It must have elemental integrity (accurate data),
vi. Access must be controlled to some degree depending on data sensitivity,
vii. A system must be in place to authenticate the users of the system, and sensitive data must be protected from inference

2.2 Conceptual Database Modeling

A model is created to represent reality, an approximation of reality, or a relevant structure. The model is then used as a substrate for analysis, database creation, or problem solving. Models are good tools because they are modifiable, scalable, and more easily constructed than a final product. Rather than focusing on the details of building the final product, the model emphasizes function and purpose while providing the basis for structural development. The basic principles of modeling have been explained, described and illustrated under conceptual database modeling, and a specific type of conceptual database modeling known as object-role modeling (ORM), has also been discussed.

Conceptual database modeling is a methodology that allows users to participate in the creation and development of databases. Conceptual database modeling is the first step in database development and is the step at which those with little or no programming experience can have the most influence on its design. A conceptual database model is like a blueprint. The graphical format of the model allows the end user to represent, using English sentences, the information needs of the application without worrying about technicalities such as programming languages, field sizes, and layout of tables. The model is in a format that end users, modelers, programmers, and information technology personnel can all understand, thereby facilitating open communication among these groups. Conceptual database modeling enables users to understand and participate in the development of information systems, thereby improving the likelihood of successful results. In object-role modeling, groups of relevant objects and roles are identified and used to create elementary facts that form the “building blocks” for information models. The resultant models can easily be communicated, reviewed, and revised, allowing decreased development time and optimizing inclusion of relevant features in the target relational database. Increasing the amount of clinical and management input in the development process may help information systems better meet user needs, become accepted and more often used, and ultimately succeed.

Conceptual database modeling is itself a process with several steps [3]; the key steps for end users are the first and second steps, describing information examples with English sentences and representing this information in a graphical model. Conceptual models portray applications at a fundamental level, using terms and concepts familiar to users. The end user need not know or complete the subsequent steps; the person serving as the modeler can finish the process. Modeling software can link all four steps and can generate a database for any database management system for which it has a driver. Although there are several different types of conceptual database modeling e.g. one that has the advantages of using natural language, intuitive diagrams, and user-specific examples. There are other structured approaches to modeling (e.g., entity-relationship modeling). Less structured approaches may serve as front ends, or interfaces, to databases.

2.2.1 Object-Role Modeling

Object-Role Modeling (ORM) is based on the premise that any process can be described in terms of a group of objects and the roles they play. The basic building block is the elementary fact, which consists of a predicate with one or more objects. An object is a person, place, or thing about which data should be collected. A predicate is the association between one or more objects. Objects play various roles, which are described by the predicate.

The first step in creating a conceptual database model with ORM is to verbalize the information required from the system, using English sentences. This method of representation is less informative and intuitive than the conceptual model style of ORM. In creating elementary facts, one is essentially identifying the objects and roles that are relevant to the desired information. Once the elementary facts have been determined, the next step is to identify how these facts fit together. Any number of objects may be interrelated, but ORM provides rules for making the descriptive sentences as simple as possible. In practice, very few sentences have more than two objects. The simplicity of the elementary facts is one of the aspects of ORM that make it accessible to the average user. These facts would be translated into additional relationships between objects.

Although there are additional details within a model that must be specified, use of modeling software makes this process accessible to the non-programming community. Although the software may include technical database terms, specification of details can be performed with simple language. For example, uniqueness constraints, which are further descriptors of the relationships between objects (ie, one-to-one, one-to-many, or many-to-one), can be determined by answering simple questions about the relationships of the objects attached to a given predicate. These answers may then be translated into a descriptive statement that determines how the constraints are placed on that particular relationship. This process also determines whether a particular role is mandatory.

After the conceptual model is complete, the modeling software checks the ORM diagrams for errors and generates the familiar logical tables that constitute the essence of a relational database. The tables are then integrated into a database management system, the backbone of an information management application. By participating in the design of the database, the end user provides valuable insights and information that enable development of a more usable information system. As implemented in modeling software, conceptual database modeling facilitates translation of database applications across different database management systems as well as reengineering of databases from existing applications. Increasing the amount of clinical and management input in the development process may help information systems better meet user needs, become accepted and more often used, and ultimately succeed.

3. Database Management System (DBMS) [11]

The Database Management System (DBMS) is the software that interacts with the users, database applications, and the database. Among other things, the DBMS allows users to insert, update, delete, and retrieve data from the database. Having a central repository for all data and the data descriptions allows the DBMS to provide a general inquiry facility to this data, called a query language. The provision of a query language (such as SQL) alleviates the problems with earlier systems where the user has to work with a fixed set of queries or where there is a proliferation of database applications, giving major software management problems. Some people use the term DBMS more generically to include functions and tools to help users develop database applications. With the functionality described above, the DBMS is an extremely powerful tool.
However, as end-users are not too interested in how complex or easy a task is for the system, it could be argued that the DBMS has made things more complex because users may now see more data than they actually need or want to do their job. In recognition of this problem, a DBMS provides another facility known as a view mechanism, which allows each user to have his or her own customized view of the database, where a view is some subset of the database.

A view is usually defined as a query that operates on the base tables to produce another virtual table. As well as reducing complexity by letting users see the data in the way they want to see it, views have several other benefits: Views provide a level of security. Views can be set up to exclude data that some users should not see. For example, we could create a view that allows a center manager and the Payroll department to see all staff data, including salary details. However, we could create a second view that excludes salary details, which other staff uses. Views provide a mechanism to customize the appearance of the database. A view can present a consistent, unchanging picture of the structure of the database, even if the underlying database is changed (for example, columns added or removed, relationships changed, data files split, restructured, or renamed). If columns are added or removed from a table, and these columns are not required by the view, the view is not affected by this change. Thus, a view helps provide additional data independence to that provided by the system.

3.1 Components of the DBMS Environment

We can identify five major components in the DBMS environment: hardware, software, data, procedures, and people:

i. Hardware: The computer system(s) that the database system runs on. This can range from a single PC, to a single mainframe, to a network of computers.
ii. Software: The DBMS software and the database applications, together with the operating system, including network software if the DBMS is being used over a network.
iii. Data: The data acts as a bridge between the hardware and software components and the human components. As we have already said, the database contains both the operational data (the data for the day-to-day running of the business) and the metadata.
iv. Procedures: The instructions and rules that govern the design and use of the database. These may include instructions on how to log on to the DBMS, make backup copies of the database, and handle hardware or software failures.
v. People: This includes the business analysts, database designers, data administrators (DAs), database administrators (DBAs), application programmers, and end-users.

3.2 DBMS architecture

DBMS architecture specifies its components (including descriptions of their functions) and their interfaces. DBMS architecture is distinct from database architecture. The following are major DBMS components:

- **DBMS external interfaces** - They are the means to communicate with the DBMS (both ways, to and from the DBMS) to perform all the operations needed for the DBMS. These can be operations on a database, or operations to operate and manage the DBMS. According to Jeffrey and Jennifer [4], an external interface can be either a user interface (e.g., typically for a database administrator), or an application programming interface (API) used for communication between an application program and the DBMS.
- **Database language engines** (or **processors**) - Most operations upon databases are performed through expression in Database languages. Languages exist for data definition, data manipulation and queries (e.g., SQL), as well as for specifying various aspects of security, and more. Language expressions are fed into a DBMS through proper interfaces. A language engine processes the language expressions (by a compiler or language interpreter) to extract the intended database operations from the expression in a way that they can be executed by the DBMS.
- **Query optimizer** - Performs query optimization on every query to choose for it the most efficient query plan (a partial order (tree) of operations) to be executed to compute the query result.
- **Database engine** - Performs the received database operations on the database objects, typically at their higher-level representation.
- **Storage engine** - translates the operations to low-level operations on the storage bits. In some references the sstorage engine is viewed as part of the database engine.
- **Transaction engine** - for correctness and reliability purposes most DBMS internal operations are performed encapsulated in transactions. Transactions can also be specified externally to the DBMS to encapsulate a group of operations. The transaction engine tracks all the transactions and manages their execution according to the transaction rules.

- **DBMS management and operation component** - Comprises many components that deal with all the DBMS management and operational aspects like performance monitoring and tuning, backup and restore, recovery from failure, security management and monitoring, database storage allocation and database storage layout monitoring, etc.

3.3 DBMS Types

Johann et al [12], writes that there are 4 main types of Database Management System (DBMS) and these are based upon their management of database structures. In other words, the types of DBMS are entirely dependent upon how the database is structured by that particular DBMS.

3.3.1Hierarchical DBMS

A DBMS is said to be hierarchical if the relationships among data in the database are established in such a way that one data item is present as the subordinate of another one. Here subordinate means that items have 'parent-child' relationships among them. Direct relationships exist between any two records that are stored consecutively. The data structure "tree" is followed by the DBMS to structure the database. No backward movement is possible / allowed in the hierarchical database. Hierarchical data model was developed by IBM in 1968 and introduced in I.M.S. (Information Management System). Conrick [13], points out that this model is like a structure of a tree with the records forming the nodes and fields forming the branches of the tree. In the hierarchical model, records are linked in the form of an organization chart. A tree structure may establish one-to-many relationship.

3.3.2Network DBMS

A DBMS is said to be a Network DBMS if the relationships among data in the database are of type many-to-many. The relationship among many-to-many appears in the form of a network. Thus the structure of a network database is extremely complicated because of these many-to-many relationships in which one record can be used as a key of the entire database. A network database is structured in the form of a graph that is also a data structure. Though the structure of such a DBMS is highly complicated however it has two basic elements i.e. records and sets to designate many-to-many relationships. Mainly high-level languages such as Pascal, COBOL and FORTRAN, etc. are used to implement the records and set structures.

3.3.3Relational DBMS

Zhuge [14], notes that DBMS is said to be a Relational DBMS or RDBMS if the database relationships are treated in the form of a table. There are three keys on relational DBMS: relation, domain and attributes. It contains fundamental constructs or records sets that contains one-to-many relationship. It is composed of rows and columns which are used to organize the database and its structure and is actually a two dimension array in the computer memory. A number of RDBMSs are available; some popular examples are Oracle, Sybase, Ingress, Informix, Microsoft SQL Server, and Microsoft Access.

3.3.4Object-oriented DBMS (OODBMS)

This is able to handle many new data types, including graphics, photographs, audio, and video. Object-oriented databases represent a significant advance over their other database cousins. Hierarchical and network databases are all designed to handle structured data; that is, data that fits nicely into fields, rows, and columns. Date [15], opines that they are useful for handling small amount of information such as names, addresses, zip codes, product numbers, and any kind of statistic or number you can think of. On the other hand, an object-oriented database can be used to store data from a variety of media sources, such as photographs and text and produce work as output, in a multimedia format. Codd [16] points out that object-oriented databases have two disadvantages. First, they are more costly to develop. Second, most organizations are reluctant to abandon or convert from those databases that they have already invested money in developing and implementing. However, the benefits to object-oriented databases are compelling. The ability to mix and match reusable objects provides incredible multimedia capability. Healthcare organizations, for example, can store, track, and recall scans, X-rays, electrocardiograms and many other forms of crucial data.

3.3.5 General-purpose DBMS

This is typically a complex software system that meets many usage requirements to properly maintain its databases which are often large and complex. This is specially the case with client-server, near-real time transactional systems, in which multiple users have access to data. Data is concurrently entered and inquired for in ways that preclude single-thread batch processing. Most of the complexities of those requirements are still present with personal, desktop-based database systems. Well known DBMSs include Oracle, FoxPro, IBM DB2, Linter, Microsoft Access, Microsoft SQL Server, MySQL, PostgreSQL and SQLite. Kroenke and David [17], states that a database is not generally portable across different DBMS, but different DBMSs can inter-operate to some degree by using standards like SQL and Oracle Database Connectivity (ODBC) together to support a single application built over more than one database. A DBMS also needs to provide effective run-time execution to properly support (e.g., in terms of performance, availability, and security) as many database end-users as needed.

3.3.6 Distributed DBMS (DDBMS) [18]

A distributed database is a collection of multiple, logically interrelated databases distributed over a computer network. A distributed database is a database that is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common central processing unit (CPU). It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers. Collections of data (e.g. in a database) can be distributed across multiple physical locations. A distributed database is distributed into separate partitions/fragments. Each

partition/fragment of a distributed database may be replicated (i.e. redundant fail-overs, RAID like).

Besides distributed database replication and fragmentation, there are many other distributed database design technologies. For example, local autonomy, synchronous and asynchronous distributed database technologies. These technologies' implementation can and does definitely depend on the needs of the business and the sensitivity/confidentiality of the data to be stored in the database, and hence the price the business is willing to spend on ensuring data security, consistency and integrity.

DDBMS has the following characteristics: (i) Horizontal fragments - subsets of tuples (rows) from a relation (table); (ii) Vertical fragments - subsets of attributes (columns) from a relation (table); (iii) Mixed fragment - a fragment which is both horizontally and vertically fragmented or a logical collection of objects in an ODBMS; (iv) Homogeneous distributed database - uses one DBMS (eg: Objectivity/DB or Oracle); (v) Heterogeneous distributed database - uses multiple DBMS's (eg: Oracle and MS-SQL and PostgreSQL).

Users access the distributed database through: (i) Local applications - Applications which do not require data from other sites; (ii) Global applications - Applications which do require data from other sites. Care with a distributed database must be taken to ensure that the following considerations:

i. The distribution is transparent: users must be able to interact with the system as if it was one logical system. This applies to the systems performance, and methods of access amongst other things.
ii. Transactions are transparent: each transaction must maintain database integrity across multiple databases. Transactions must also be divided into sub-transactions, each sub-transaction affecting one database system.

The distributed database has all of the security concerns of a single-site database plus several additional problem areas. In developing a distributed database, one of the first questions to answer is where to grant system access. Beynon [10], outlines two strategies: The first case is access at home. This is no more difficult to implement than a centralized access strategy. Beynon pointed out that the success of this strategy depends on reliable communication between the different sites (the remote site must receive all of the necessary clearance information). Since many different sites can grant access, the probability of unauthorized access increases. Once one site has been compromised, the entire system is compromised. If each site maintains access control for all users, the impact of the compromise of a single site is reduced (provided that the intrusion is not the result of a stolen password).

The second strategy – centralized access, while perhaps more secure, has several disadvantages. Probably the most glaring is the additional processing overhead required, particularly if the given operation requires the participation of several sites. Furthermore, the maintenance of replicated clearance tables is computationally expensive and more prone to error. Finally, the replication of passwords, even though they're encrypted, increases the risk of theft. A third possibility offered by Date, Hugh and Nikos [19], centralizes the granting of access privileges at nodes called policy servers. These servers are arranged in a network. When a policy server receives a request for access, all members of the network determine whether to authorize the access of the user. Woo and Lam believes that separating the approval system from the application interface reduces the probability of compromise.

According to Beynon [10], preservation of integrity is much more difficult in a heterogeneous distributed database than in a homogeneous one. The degree of central control dictates the level of difficulty with integrity constraints (integrity constraints enforce the rules of the individual organization). The homogeneous distributed database has strong central control and has identical DBMS schema. If the nodes in the distributed network are heterogeneous (the DBMS schema and the associated organizations are dissimilar), several problems can arise that will threaten the integrity of the distributed data. They list three problem areas:

i. Inconsistencies between local integrity constraints,
ii. Difficulties in specifying global integrity constraints,
iii. Inconsistencies between local and global constraints

Beynon also explain that local integrity constraints are bound to differ in a heterogeneous distributed database. The differences stem from differences in the individual organizations. These inconsistencies can cause problems, particularly with complex queries that rely on more than one database. Development of global integrity constraints can eliminate conflicts between individual databases. Yet these are not always easy to implement .Global integrity constraints on the other hand are separated from the individual organizations. It may not always be practical to change the organizational structure in order to make the distributed database consistent. Ultimately, this will lead to inconsistencies between local and global constraints. Conflict resolution depends on the level of central control. If there is strong global control, the global integrity constraints will take precedence. If central control is weak, local integrity constraints will take precedence.

3.4 Roles of DBMS

DBMS stands for Database Management System. So its role is basically to manage the database. More specifically this software controls the storage, organization, retrieval, integrity and security of the data in the database. Codd [20], opines that without a database management system organizing, controlling and cataloging data, an information system would be an organized conglomeration of data. The ultimate role of a database management system is to implement controls and provide maintenance to data files using data security to ensure integrity of data. The process of

cataloging files in a DBMS is extremely important. There are various file types, which range from actual computer code and query programs (which extract information) to system utility and record maintenance programs. All of these programs have a unique file structure, which is identified by a system schematic or "schema." Without the process of a file structure, files would be hard to access and operate. A file structure within a DBMS provides an orderly structure for file access and management. These roles include support for all of the following:

- Data definition: The DBMS must be able to accept data definitions (external schemas, the conceptual schema, the internal schema, and all associated mappings) in source form and convert them to the appropriate object form.
- Data manipu1ation: The DBMS must be able to handle requests from the users to retrieve, update, or delete existing data in the database, or to add new data to the database. In other words, the DBMS must include a data manipulation language (DML) processor component.
- Data security and integrity: The DBMS must monitor user requests and reject any attempt to violate the security and integrity rules defined by the Database Administrator (DBA).
- Data recovery and concurrency: The DBMS - or else some other related software component, usually called the transaction manager - must enforce certain recovery and concurrency controls.
- Data Dictionary: The DBMS must provide a data dictionary function. The data dictionary can be regarded as a database in its own right (but a system database, rather than a user database). The dictionary contains “data about the data” (sometimes called metadata) - that is, definitions of other objects in the system - rather than just raw data. In particular, all the various schemas and mapping (external, conceptual, etc.) will physically be stored, in both source and object form in the dictionary. Lightstone [21] notes that a comprehensive dictionary will also include cross-reference information, showing, for instance, which programs use which pieces of the database, which users require which reports, which terminals are connected to the system, and so on. The dictionary might even (in fact, probably should) be integrated into the database it defines, and thus include its own definition. It should certainly be possible to query the dictionary just like any other database, so that, for example, it is possible to tell which programs and or users are likely to be affected by some proposed change to the system.
- Performance: It goes without saying that the DBMS should perform all of the roles identified above as efficiently as possible.

3.5 DBMS Implementation

After designing a database for an application arrives the stage of building the database. Typically an appropriate general-purpose DBMS can be selected to be utilized for this purpose. A DBMS provides the needed user interfaces to be utilized by database administrators to define the needed application's data structures within the DBMS's respective data model. Other user interfaces are used to select needed DBMS parameters (like security related, storage allocation parameters, etc.). When the database is ready (all its data structures and other needed components are defined), it is typically populated with initial application's data (database initialization, which is typically a distinct project; in many cases using specialized DBMS interfaces that support bulk insertion) before making it operational. In some cases the database becomes operational while empty from application's data, and data are accumulated along its operation. After completing, building the database and making it operational arrives the database maintenance stage: various database parameters may need changes and tuning for better performance, application's data structures may be changed or added, new related application programs may be written to add to the application's functionality, etc.

Codd [20] also states that a database built with one DBMS is not portable to another DBMS (i.e., the other DBMS cannot run it). However, in some situations it is desirable to move, migrate a database from one DBMS to another. The reasons are primarily economical (different DBMSs may have different total costs of ownership or TCOs), functional, and operational (different DBMSs may have different capabilities). The migration involves the database's transformation from one DBMS type to another. The transformation should maintain (if possible) the database related application (i.e., all related application programs) intact. Thus, the database's conceptual and external architectural levels should be maintained in the transformation. It may be desired that also some aspects of the architecture internal level are maintained. A complex or large database migration may be a complicated and costly (one-time) project by itself, which should be factored into the decision to migrate. This in spite of the fact that tools may exist to help migration between specific DBMS. Typically a DBMS vendor provides tools to help importing databases from other popular DBMSs.

A typical DBMS cannot store the data of the application it serves alone. In order to handle the application data the DBMS need to store these data in data structures that comprise specific data by themselves. In addition the DBMS needs its own data structures and many types of bookkeeping data like indexes and logs. The DBMS data are an integral part of the database and may comprise a substantial portion of it. Since a database management system (DBMS) is a system that allows to build and maintain databases, as well as to utilize their data and retrieve information from it. It implements solutions to data and database usability requirements. It defines the database type that it supports, as well as its functionality and operational capabilities. Chapple [22], notes that a DBMS provides the internal processes for external applications built on them. The end-users of some such specific application are usually exposed only to that application and do not directly interact with the DBMS. Thus end-users enjoy the effects of the underlying DBMS, but its internals are completely invisible to end-users. Database designers and database administrators interact with the DBMS through dedicated interfaces to build and

maintain the applications' databases, and thus need some more knowledge and understanding about how DBMSs operate and the DBMSs' external interfaces and tuning parameters.

Teory, Lightstone, and Nadeau [23] writes that a DBMS consists of software that operates databases, providing storage, access, security, backup and other facilities to meet needed requirements. DBMSs can be categorized according to the database model(s) that they support, such as relational or XML, the type(s) of computer they support, such as a server cluster or a mobile phone, the query language(s) that access the database, such as SQL or XQuery, performance trade-offs, such as maximum scale or maximum speed or others. Some DBMSs cover more than one entry in these categories, e.g., supporting multiple query languages. Database software typically support the Open Database Connectivity (ODBC) standard which allows the database to integrate (to some extent) with other databases. The development of a mature general-purpose DBMS typically takes several years and many man-years. Developers of DBMS typically update their product to follow and take advantage of progress in computer and storage technologies. Several DBMS products like Oracle and IBM DB2 have been in on-going development since the 1970s-1980s.

4. Logical and Physical Database Organizations

Database tables/indexes are typically stored in memory or on hard disk in one of many forms, ordered/unordered Flat files, ISAM, Heaps, Hash buckets or B+ Trees. The most commonly used are B+ trees and ISAM [24]. Other important design choices relate to the clustering of data by category (such as grouping data by month, or location), creating pre-computed views known as materialized views, partitioning data by range or hash. As well memory management and storage topology can be important design choices for database designers. Just as normalization is used to reduce storage requirements and improve the extensibility of the database, conversely de-normalization is often used to reduce join complexity and reduce execution time for queries.

4.1 Sequential File Organization

The physical order of sectors, tracks, and cylinders in which blocks are written, (and therefore subsequently read) is defined so as to minimize access times. This means that all sectors within the same cylinder are written to before moving to the next cylinder so as to minimize head movement. (Working from surface to surface on the same cylinder does not require movement of the read-write head). It also means that the sectors within a track are written in an order that reduces rotational delay. Ideally, this would mean that the sectors are written (and read) in numbered sequence 1, 2, 3, etc but normally delays in the software or hardware controlling the reads and write means that one or more sectors have to be skipped between writes. For example, if there are 8 sectors on a track, the order of reads might be 1, 4, 7, 2, 5, 8, 3, 6 with a delay of two sectors between each read.

4.1.1 Index Sequential Organization

All of these databases can take advantage of indexing to increase their speed, and this technology has advanced tremendously since its early uses in the 1960s and 1970s. The most common kind of index is a sorted list of the contents of some particular table column, with pointers to the row associated with the value. An index allows a set of table rows matching some criterion to be located quickly. Typically, indexes are also stored in the various forms of data-structure mentioned above (such as B-trees, hashes, and linked lists). Usually, a specific technique is chosen by the database designer to increase efficiency in the particular case of the type of index required.

Relational DBMSs have the advantage that indexes can be created or dropped without changing existing applications making use of it. The database chooses between many different strategies based on which one it estimates will run the fastest. In other words, indexes are transparent to the application or end-user querying the database; while they affect performance, any SQL command will run with or without indexes existing in the database.

Relational DBMSs utilize many different algorithms to compute the result of an SQL statement. The RDBMS will produce a plan of how to execute the query, which is generated by analyzing the run times of the different algorithms and selecting the quickest. Some of the key algorithms that deal with joins are Nested loop join, Sort-Merge Join and Hash Join. Which of these is chosen depends on whether an index exists, what type it is, and its cardinality [25].

The same principles apply with respect to minimizing access time but the situation is complicated by the more complex organization of the file. The index sequential file is created from a file that is already in sequential order. The indexes are generated and included as the index sequential file is organized and stored. The indexes are subsequently updated as the file is updated.

i. Primary Index: This is created in main storage as the file i.e. organized, and stored on the disk when organization and storage of the file is completed. It is loaded into main storage again at the start of any subsequent access. The primary index is normally organized as a sequential file on its own area of the disk e.g. on its own cylinder.

ii. Secondary Index: This is also created in main storage while each cylinder is organized and stored. There is one secondary index per cylinder. During the organization of each cylinder provision is made for local overflow, i.e. the provision of spare storage on the same cylinder, for tracks that subsequently may become full and therefore unable to accommodate further records. (Global overflow, i.e. overflow of the whole file, may be catered for in a similar way.)

Space is left on each surface during the initial organization so that a few additions can be made before overflow occurs. When the file is accessed, in the absence of any overflow, the access times for primary index, secondary index and the data access itself are kept to a minimum. Performance degrades as the amount of overflow increases because of the additional index references incurred.

4.2 Random File Organization

In this method the key are used to allocate record positions on the disc. For example, a record whose key was 149 could be allocated the position surface 1 track 49. We say that the disk address of the record has been generated from the key and so the technique is called address generation. The generated disk address usually gives just enough detail to specify the block in which the record is to be placed or found; so that when a record is to be accessed the whole block is input, and the record is searched for within the block. We thus have organization by address generation and access by address generation. Sometimes an index of generated addresses is produced as the file is created. This index is then stored with the file. It is then possible to access the file by means of this random index. We then have organization by address generation and access by random index.

4.2.1 Hashed Keys

When disk addresses are generated directly from keys, as in the example just given, there tends to be an uneven distribution of records over available tracks. This can be avoided by applying some algorithm to the key first. In this case we say the key is hashed. Examples:

a. Squaring, e.g. for key number 188

188^2	35	3	4	4
DISC ADDRESS	Track Number	Surface Number	Bucket Number	Block Number

b. Division method, e.g. for key number 188, is 188 /7 = 26 Remainder 6. So we could use track 26 surface 6 say.

Hashing reduces the chances of overflow occurring, but when overflow does occur records are normally placed on the next available surface in the same cylinder so as to minimize head movement.

We might ask if it is possible to create a data structure that does not require a search to implement the find operation. Is it possible for example to compute the location of the record that has a given key value: Memory address of records = f (key), Where, f is a function that maps each distinct key value into the memory address of the records identified by the key. Such function can be found, but they are difficult to determine and can only be constructed if all of the keys in the data set are known in advance. They are called perfect hashing functions. This function does not necessarily give the exact memory address of the target record but only give a home address that many contain the desired record: Home address = H (key)

4.2.2 Rehashing

Function such as H are known as hashing function. They are easy to determine and can give excellent performance. Although the home address many not contain the record being sought. In that case, a search of other addresses may be required. This is known as rehashing. The fundamental idea behind hashing in the antithesis (the direct opposite) is sorting. A sort arranges the records in a regular pattern that makes the relatively efficient binary search possible. Hashing takes diametrically opposite approach. The basic idea in hashing is to scatter the records completely throughout some memory or storage space.

The hash function can be thought of as a pseudo-random number generation that uses the value of the key as a seed and that outputs the home address of the element containing the key. It allows us to find records with 0 (1) probes. Let us imagine the hash function H is: H(key) = key mod 7
Please observe that the value produced by this function is always an integer between 0 and 6 which is within the range of indexes of the table.

One of the drawbacks is the random (scatter) location of stored element. There is no notion of first, next, parent or child or anything.

4.2.3 Collision

Can be described as when two different key value hashing to the same location. Why thus happens and what to do about it are important because collisions are a fact of life when hashing for example: H (227) = 227 mod 7 = 3

The birthday paradox described that the probability that there is no collisions is essentially zero. Codd, [20], explained that hash function with no collision is rare and it is worth looking for them only in very special circumstances.

Strategies for handing collisions are normally called rehashing or collision resolution strategies. This determines what happen when two or where elements have a collision or hash to the same address. There are three strategies or approaches to resolve collision. They are as follows:

i. Open Address method: This method attempts to place a second and subsequent keys that hash to the one table location into some other position in the table that is unoccupied (open) e.g. in our earlier example.

   H (227) = 227 mod 7 = 3. Therefore 227 collide with 374. A simple resolution to the collision called linear rehashing is to start a sequential search through the hash table at the position at which the collision occurred. The search continues until an open position is found or until the table is exhausted.

ii. External chaining: This is a second approach to the problem of collision. In this approach the table positions absorb all the records that hash to it. Since we do not usually know how many keys will hash into any table

position, a linked list is a good data structure to collect the record. H (key) = key mod 7

External Chaining has three advantages over open address method.

- Deletions are possible with no resulting problems.
- The number of elements in the table can be greater than the table size; this can be greater than 1.0. Storage for the elements is dynamically allocated as the lists grow larger.
- That the performance of external chaining in executing a find a key operation is better than that of open address methods and continues to be excellent as it grows beyond 1.0.
- In the next technique collisions are resolves, as they are in external chaining, by adding the element to be inserted to the end of a list. The different is in how the list is constructed.

iii. Coalesced chaining: coalesced chaining behaves exactly like external chaining – each new record is added to the end of a list that begins at its home address, except that the next insertion illustrates how a collision is resolved after the cellar is full; Such that, once again the record being inserted was, since its home address was already occupied, placed in the empty position with the largest address.

4.3 Other Organizations and Access Methods

The methods of file organization and access described so far are closely related to the physical features of the disk. It is now common practice for much of this detail to be "hidden" from programmers or users of the system. Instead of operating system handles the physical levels of organization and access, and provides standard logical file organization and access method. Some examples of logical file organization include:

i. Sequential Files: A programmer need merely regard the file as a sequence of records, and need have no concern for their physical location. A program instruction to read the next record will result in the appropriate record being transferring into memory, i.e. the programmer's "view" may just be like this.

R1	R2	R3	R4	R5	R6	etc

R1…R6 are logical records.

ii. Direct Files: There are files that provide fast and efficient direct access i.e. they are normally random files with one of a number of appropriate addressing methods. A common type of direct file is the Relative File. The logical organization of a relative file is like this:

R1	R2	R3	R4	R5	R6 etc

R1…R6 are logical records with logical keys 1…6
A relative file may be accessed sequentially or randomly.

iii. Index Sequential Files: Logical versions of index sequential files are simpler than their physical counterparts, in that logical keys are used instead of disk addresses, and details of overflow are hidden from the programmer.

4.3.1 Choice of File Organization and Access Methods
There are several characterizes factors that determined the type of file organization and access to be applied at any particular time. The factors affecting choice includes:

i. Size: Disks and tapes are both capable of storing very large files but very large files are stored more economically on magnetic tape. Small files can use disk space well if blocks are suitably sized and can be easily accessed, but may be troublesome to access if strung (combination of bits) along magnetic tape because of the serial nature of tape. The percentage of additions or deletions in file maintenance if low many allow satisfactory organization of indexed or random files, but if high will make sequential organization more attractive. If a file is like to increase in size then the design of an indexed or random file must allow for it.

ii. Reference and Enquiry: If quick reference is essential then indexed or random files will be needed. If the file is only used for quick one-off reference then a random organization may be best.

iii. Hit Rate: Low hit rates are suited by indexed or random files, and high hit rates are suited by sequential files. Putting together batches of enquiries to form a transaction file can raise hit rates.

iv. Security and Backup: The "Father-Son" concept is an aid to security when using sequential files. Indexed and random files are overwritten during processing so they may need to be dumped onto tape or another disk between processing, as well as keeping a copy of all transaction between dumps.

4.4 File Calculations
Two basic types of calculation are often needed when using files:

a. The storage space occupied by the file (for magnetic tape the length of tape may be required).
b. The time taken to read or write the file.

A simple example now follows.

For a sequential file on disk the basic calculation for estimating the required space is as follows:

- Divide the block size by the record size to find how many whole records can fit into a block. This is the blocking factor.
- Divide the total number of records by the blocking factor to obtain the total number of blocks required.

- Multiply the block size in bytes by total number of blocks required.

This basic method can be modified if the records are variable in length (e.g. use an average).

For non-sequential files on disk, the storage space required is greater than that for sequential because of the space allowed for insertions and overflow. The exact calculations depend on the software used to organize the files and on the ways in which it is possible to configure the settings. However, a typical overhead is 20% more than that for sequential. The basic calculations of read and write times on disk are as follows.

Average access time = seek time + latency + data transfer time.

For a random file where N records were read the total time would simple be: **N x average access time.**

Please take note that a whole sector would have to be read in just to get each individual record. For a sequential file, where the disk was not being used for any other accesses at the same time there would be a seek for each cylinder and then all sectors in the cylinder would be read. This suggests a formula such as:

Total read time = seek time x latency + data x sectors per x cylinder transfer time per file

4.5 Other Types of Storage Methods

Having examined file organization and access we are now in a better position to review files storage methods. The main types of storage are:

i. Immediate-access storage (IAS) e.g. RAM - Because of its unique electronic properties giving extremely quick access to stored this type of storage is used as the computer's main storage. Access to individual characters/bytes is completely independent of their position in store. It is in main storage that the programs are held during processing so that their instructions can be executed swiftly. Also the particular data currently being worked on is held in main storage. Ideally, IAS would be used for storage of all data because it is fast in operation. This is not practicable because of cost and volatility, and its use therefore is limited to main storage. Thus some alternative form must be found for storing the files and data, which are not immediately required by the program.

ii. Direct-access storage (DAS) - One such alternative is DAS (e.g. disk storage). Storage capacity can vary from thousands to hundreds of millions of characters. DAS has the important facility of allowing direct access, that is, records can be accessed independently of each other. It is thus suitable for files being processes in a selective manner and also for storing programs, and systems software that are required to be called into main storage at any time during the running of an application program. It is an essential requirement for on-line processing or where file-interrogation facilities are needed.

iii. Serial-access storage (SAS) e.g. magnetic tape - If bulky auxiliary storage is required and the need for random access is not present then the choice may well fall on SAS, the most common form of which is magnetic tape. It is also cheaper than main memory or disk. Because of its inherent serial nature, access to a record on tape is a function of its position on the tape. Therefore very preceding record on a master file must be read (and written out onto the new tape) before the required record can be located. Nevertheless millions of characters can be stored very cheaply.

5. Conclusion

Databases are used in many applications, spanning virtually the entire range of computer software. Databases are the preferred method of storage for large multiuser applications, where coordination between many users is needed. Even individual users find them convenient, and many electronic mail programs and personal organizers are based on standard database technologies. This study has been carried out to show that database technology concepts, implementations and organizations have developed over the years and will yet develop, and the all the stages of improvements till date can be traced for academic, and other purposes.

References

[1] Gehani, N., "The Database Book: Principles and practice using MySQL". 1st ed., Summit, NJ.: Silicon Press, (2006).
[2] Berkowitz L. L., "Diagnosing doctors: four types of physicians require four approaches to promote clinical computing acceptance" Healthc Inform; 15:93-96, (1998).
[3] Hawkins H. Hugh, Young Scott K., Hubert C. Katherine, Hallock Patrick, "Conceptual Database Modeling for Understanding and Developing Information Management Applications", (2007).
[4] Connolly, T. and Carolyn, B., Database Systems. New York: Harlow, (2002).
[5] Date, C. J., "When's an extension not an extension?". IntelligentEnterprise. Available: http://intelligent-enterprise.informationweek.com/db_area/archives/1999/990106/online1.jhtml;session, [Accessed: Dec 12, 2013].
[6] Codd, E.F., "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–387. doi:10.1145/362384.362685. Available: http://www.acm.org/classics/nov95/toc.html. [Accessed: Dec 15, 2013].
[7] Date, C. J., Database in Depth: Relational Theory for Practitioners. p. 152, (2005).
[8] Ling, L. and Tamer, M. Ö (Eds.), "Encyclopedia of Database Systems, 4100 p. 60, (2009).

[9] Beynon,D P., Database Systems 3rd Edition. Palgrave, Basingstoke, UK, (2004).
[10] Halpin T. A., "Conceptual schema and relational database design", 2nd ed. Sydney, Australia: Prentice Hall, (1995).
[11] Jeffrey, U. and Jennifer, W., First course in database systems, Prentice-Hall Inc., Simon & Schuster, (1997).
[12] Johann, A.; Makowsky, V. & Nimrod R., "Entity-relationship consistency for relational schemas" Proceedings of the 1986 Conference on Database Theory (ICDT '86), Lecture Notes in Computer Science, 1986, Volume 243/1986, pp. 306-322, Springer, doi:10.1007/3-540-17187-8_43, (1986).
[13] Conrick, M., "Introducing databases, health informatics: transforming healthcare with technology, Thomson, ISBN 0-17-012731-1, p. 69, (2006).
[14] Zhuge, H., The Web Resource Space Model. Web Information Systems Engineering and Internet Technologies Book Series4. Springer, (2008).
[15] Date, C. J., An Introduction to Database Systems, Fifth Edition. Addison Wesley, (2003).
[16] Codd, E. F., "Recent Investigations into Relational Data Base Systems". IBM Research Report RJ1385. Republished in Proc. 1974 Congress (Stockholm, Sweden, New York, N.Y.: North-Holland, (1974).
[17] Kroenke, D. M. and David, J. A., Database Concepts. 3rd ed. New York: Prentice, (2007).
[18] Osuagwu O. E., "Distributed Intelligent Data Base Networks, Concurrent & Real Time Communications", First Edition, Alphabet Publishers, Owerri, pp 100-150, (2010).
[19] Date, Chris J.; Hugh Darwen, Nikos A. Lorentzos, "Chapter 10: Database Design - Temporal Data and the Relational Model: A Detailed Investigation into the Application of Interval and Relation Theory to the Problem of Temporal Database Management". Oxford: Elsevier LTD, p. 176. ISBN: 1558608559 (2003).
[20] Codd, E.F., "Serious Flaws in SQL", in The Relational Model for Database Management: Version 2. Addison-Wesley, pp. 371–389, (1990).
[21] Lightstone S., Teorey T., Nadeau T., "Physical Database Design: the database professional's guide to exploiting indexes, views, storage, and more", Morgan Kaufmann Press, ISBN 0-12-369389-6, (2007).
[22] Chapple, M. "SQL Fundamentals". Databases. About.com. Available: http://databases.about.com/od/sql/a/sqlfundamentals.htm. [Accessed: December 12, 2013]
[23] Teorey, T.; Lightstone, S. and Nadeau, T., Database Modeling & Design: Logical Design, 4th edition, Morgan Kaufmann Press. ISBN 0-12-685352-5, (2005).
[24] Galindo, J., Urrutia, A., Piattini, M., "Fuzzy Databases: Modeling, Design and Implementation" (FSQL guide). Idea Group Publishing Hershey, USA, (2006).
[25] Galindo, J., "Handbook on Fuzzy Information Processing in Databases", Information Science Reference (an imprint of Idea Group Inc.), Hershey, USA, (2008).

Authors Profile

Udoka Felista Eze (MCPN, MNCS, MISOC, MNITAD, MNIWIIT) received the B.SC, M.SC and Ph.D degrees in Computer Science from Nnamdi Azikiwe University, Awka. She is a lecturer in the department of Information Management Technology, FUTO. Her research areas include: Computer Systems, Database Systems, Electronic and Computer system, Artificial Intelligence, and Information Systems.

Chukuemeka Etus received the B.Eng and M. Eng. degrees in Electrical/Electronic Engineering from Federal University of Technology, Owerri in 2005 and 2010, respectively. He is a lecturer in the department of Information Management Technology, FUTO. His research areas include: Electronic and Computer system, Smart Environments and Working Smart, Artificial Intelligence, Telecommunication systems and Information Technology.

Joy Ebere Uzukwu received the B.SC, M.SC, and PGDE (education) degrees in Computer Science and Computer Education from Nnamdi Azikiwe University, Awka, and National Teachers Institute, Awka respectively in 2005 and 2010, respectively. She is a lecturer in the department of Computer Science Federal Polytechnic, Oko. She has authored several books and publications in the field of computing. She is also a conference speaker and motivational writer.