International Journal of Innovative Research and Advanced Studies (IJIRAS)
Volume 3 Issue 8, July 2016

ISSN: 2394-4404

# A Graph Model Object Oriented Database For Data Mining

**Onu Fergus U.**

**Anujeonye Nneamaka N.**

**Nwanze Maureen N.**

Department of Computer Science,
Ebonyi State University, Abakaliki – Nigeria

***Abstract:** The increased complexity of software as users clamors for more user-defined functionality, now compels database modelers to refocus their design problems. The phenomenon caused a paradigm shift from just database design to the overall architecture of the database. Through a critical and detailed study of materials from secondary sources, this paper presented the use of relational database models extended with object-oriented concepts to yield a hybrid graph model and as well the use of descriptive mining technique for large data, so that data are appropriately classified and displayed on demand during a search. With this technique, software developers, database designers, programmers and software/database architects would achieve robust, flexible and adaptable model frameworks for software projects. These have been reckoned with as the key factors for the success or consequently, failure of many software projects.*

***Keywords:** Database Model, Object Oriented Concepts, Graph Model, Data Mining, Relational Data Model*

## I. INTRODUCTION

STORAGE Optimization techniques, virtualization and BIG data in recent times became imperative with the need to aid proper and effective storage for large amount of data generated today. As researchers and software-designers sought better techniques to achieve these, the exponential, continued rapid growth of data in various types/forms today – beckons on the need for efficient frameworks that proffers dependable solutions to curb such data explosion and help store such large amounts of data (Bloom and Zdonik, 1987). Data compression techniques, has its corresponding issues that made possible the pursuance of more efficient means to curb such growth. But, data is stored mainly via one-of-two means: file-based system and database system (Satesh and Patel, 2009).

This study was motivated by:

- ✓ users need to extend Relational Database Management System (RDBMS) so as to capture variants of data types,
- ✓ search and classification of data in large databases can be burdensome as users are often stuck with the option of what database model and design to use in their object oriented programming (OOP) application,
- ✓ user seek better means to implement in their application in the event that a drastic change in data type(s) originally modeled from outset will automatically demand a corresponding change in database structure, model used and implemented by the application at run time, and
- ✓ difficulties involved in supporting an OOP concepts via an OOP application with RDBMS.

First issue is resolved by seeking an appropriate database model to use, and for this study, we have sought the use of relational database extended with object-oriented concepts to yield a hybrid (graph) model (as in section II). Also the search for data in very large databases (or as the database grows) can be quite frustrating and a herculean task. Thus, we model the use of descriptive mining technique for large data, so that data are appropriately classified and displayed on demand (as resolved in Section II). The issue of what database schema and query to implement by organizations (and other users) in their application so that a drastic change in the data type and format in use, will not automatically demand a corresponding change in the database structure and model used. To resolve this, the study advances the framework as proposed in Section III.

## II. LITERATURE REVIEW

### A. DATABASE AND DATA STORAGE

A permanent way to store data on a computer is via files. An organisation uses various applications, each manipulates data in files and in various formats. Each file stored (irrespective of its name and extension) is achieved via a *file-based system*, which allows data manipulated to be stored in a file. The issues inherent in the file-system includes: integrity, isolation, redundancy, security, format inconsistency and concurrency access issues (Watt and Eng, 2013). These difficulties arising from a single file-based system has necessitated the need for development and growth of a new approach to store data; Thus, the development of a new model called the database model (Bloom and Zdonik, 1987).

A database is a shared collection of related data that support the activities of a particular organization. It is a repository of data, which defined once – is accessed by various users. Its features include: (a) it represents aspects of, or a collection of elements (facts) of real-world data, (b) it is logical, coherent and consistent, (c) it is built, designed and populated with data for a specific task, (d) each item is stored in a field, which are populated to make up a record, and (f) a combination of fields (and records) define its table, and the table-structure. Thus, we redefine the database as a system that contain table(s) that are housed within a database management system (DBMS). The DBMS is a collection of programs that enable users to create, maintain and utilize full controlled access to databases. The primary goal of a DBMS is to provide an environment that is both convenient and efficient for users to manipulate, retrieve and store information (Watt and Eng, 2013; Ojugo et al,).

DBMS provides one/more models whose optimal structure on which it stores and manipulates data, depends on its natural organization requirement and application's data, which include reliability, rate of transaction, maintainability, scalability and cost (Codd, 1970). DBMS are built around a data-model and it is possible for DBMS to offer support for more than a model. Also, most DBMS offer users some level of control to tune its physical structure – since these choices when made, creates a significant effect on its performance (Conrick, 2006).

### B. DATA MODELS

A database model determines the logical structure of how its physical data is stored, organized and manipulated on storage (primary and secondary) media. It defines the set of operations that can be performed on the data. For example, the relational model (a popular data models) defines operation such as select (project) and join. Though, the operations may not be explicit in a particular query language, they are foundations on which a query language is built. There are various logical database models to include (Robie and Bartel, 2013; Conrick, 2006):

- ✓ Hierarchical Model organizes data as tree-structure with single parent for each record. It was widely used in early mainframe IBM Information Management System; But, now describes the XML structure. It allows one-to-many relations of two data types, and efficient to describe many relationships such as table of contents, nested loops, sorted data and ordering of paragraphs. The hierarchy is used as physical order of records in storage. Access to record is by navigating downward via pointers combined with sequential access. This makes it inefficient for some database operations when full path (as opposed to upward link and sort field) is not included for each record. Such limitations were compensated for in later IMS versions by additional logical hierarchies imposed on its base physical hierarchy (Zhuge, 2008; Date, 1999).
- ✓ Network Model extends the hierarchical model via many-to-many relation of multiple parents to define CODASYL specification. It organizes data via two concepts: records and sets. Records have fields hierarchically organized as in COBOL; And *sets* define record one-to-many relation: one owner, many members. A record may be an owner in a number of sets, and a member can be in any number of sets. A set consists of circular linked lists such that one record type (a set owner or parent) can appear once in each circle; while, a second record type (its subordinate or child) can appear multiple times in each circle. This establishes hierarchy between any two record types. Thus all sets comprise a general directed-graph (ownership defines direction) or network construct. Access to records is either sequential (in each record type), or by navigation in circular linked lists. This model represents redundancy in data efficiently than hierarchical model. There can be more than one path from an ancestor to a descendant. The program maintains a current position as it navigates from one record to another following all relations the record participates in. Records are located via key values. Model uses set relations of pointers that directly address location of a record. It yields excellent retrieval performance at the expense of other operations such as database loading and reorganization. Example is Cullinet's IDMS (Codd, 1970).
- ✓ Relational Model – Codd (1970) introduced it as a mathematical model to make DBMS more independent of particular application via predicate logic and set theory. It is based on 3-key feats: relations, attributes, and domains. A relation is a table with columns (called attributes) and rows. Domain is set of values its attributes are allowed to take. Its uses a table: data about an entity (e.g., employee record) is presented in its rows and columns. The columns are various attributes of an entity such as employee name, address or phone number etc; while, row is an instance of the entity (specific employee) – to represent the relation. The 'relation' refers to various tables. Each tuple in the table represents various attributes of an employee; And all relations (tables) must adhere to basic rules that qualify it as relations, which includes: (a) ordering of the column is immaterial, (b) there cannot be identical rows in a table, and (c) each tuple contains single value for each attribute (Codd, 1970). Its strength is that, a value input in two different records (of same or different tables) is a relation between the two records. To enforce integrity, relations among records are defined by identifying a parent-child relation characterized via cardinality (1:1, (0)1:M, M:M).
- ✓ Tables can have single or set of attribute keys to uniquely identify each tuple in a table (Zhuge, 2008). A primary

key uniquely identifies a row in a table and is commonly used to join data from two/more tables. Keys help with indexing to facilitate fast retrieval of data from large tables. A column can be a key, or multiple columns are grouped together into compound key. Keys can be defined at any time. A key that has an external, real-world meaning (person's name, book ISBN, or serial number) is called a natural key. In practice, most databases have both generated and natural keys (generated keys can be used internally to create links between rows that cannot break, while natural keys are used for searches and integration with other databases). Most common query language with RDB is Structured Query Language (Okonta et al, 2015).

✓ Object-Oriented Model – uses the concept of objects to avoid object-relational impedance mismatch, an overhead of converting data between its representation in database (as rows in tables) and its representation in the application program (objects); Or with data-types used in a particular application that is directly defined in database that allows database to enforce same data integrity invariants. Object database (OODB) adapt encapsulation and polymorphism into databases by allowing objects that are manipulated by the program to be persistent via addition of queries (since traditional programming languages do not have the ability to find objects based on their data content). OODB model is based on objects with a structure that combine related code and data – as defined in its class declaration. The basic feats of objects are: (a) encapsulation allows codes and data to be packed together as an object so that its implementation is hidden from program, (b) inheritance allows new class to be built using code and data declared in another, and it allows common feats of a set of classes to be expressed in its base class, and code of related class, and lastly, (c) polymorphism allows us to create instances of an object from its class. Each object has its identity, which does not depend on values it contains. The object's address is its identity, and pointer references are then used to establish relations among objects so that its container classes are created as many-to-one relation (Robie and Bartels, 2013). OODB define a database via its language and allow full programming capabilities and traditional query facilities to be available to users. It suffers standardization as it has not been used well enough to ensure interoperability. OODB success are in many applications such as engineering and molecular biology; Rather, than commercial purpose. An alternative to translate between OODB and RDB is the use of object-relational mapping (ORM) library (Bagui, 2003).

## C. RELATIONAL VERSUS OBJECT-ORIENTED DBMS

Databases, regardless of structure are problem-specific and aim to offer a uniform framework for what is now an accepted solution for efficient storage and retrieval of large volumes of data (Fong, 1997). Many formats have been presented; But, RDBMS has been the most adopted over time. It uses a table-based structure for static components to organize data, and can handle simple predefined data-types. Issue(s) however, arise when it needs to deal with complex data-types, user-defined data and/or multimedia data (Leavit, 2000) – which implies that RDBMS fails to handle complex data systems (Alam and Wasan, 2006) as its semantics are left unexplored within many relationships that cannot be extracted without the users' help (Satesh and Patel, 2009).

In an attempt to use RDBMS technology in data processing activities like computer aided manufacturing and design, web-mining, knowledge-based systems, software engineering and multimedia systems – OODBMSs were adopted by modelers and database czars in many real-time applications to evade such shortcomings in RDBMS (Bagui, 2003). This has further imploded a paradigm shift from RDBMS to OODBMS, and finally to agent-based database systems. This shift has been necessitated by the need to perform complex manipulations on database systems yielding a new generation of hybrid database systems that use one or more concepts with another generation of database system whose requirement cum heuristic is better satisfied by OODBMS and such hybrids (Satesh and Patel, 2009). To the rescue, is Object-Oriented database (OODBMS) – whose structure is based on the concept of objects using fets such abstraction, inheritance, polymorphism etc. These allow it to use the abovementioned object-model to capture the many complexities and semantics of data (Fong, 1997). Studies and organizations are now implementing OODBMS as means to resolve issues of data storage, retrieval and processing (Nunn-Clark et al, 2003). The major strength of OODB is its ability to handle applications with complex, interrelated data (Alam and Wasan, 2006). Object-oriented DBMSs were necessitated by defects in the RDBMS and its inability to meet processing requirements in some real-time applications.

## D. THE HYBRID DATABASE FRAMEWORK

There are cases for which OODBMS are quite inefficient to compete in the same task as its RDBMS counterparts. Various applications have been designed around RDBMS and studies prove that it is quite difficult (if not impossible) to move off RDBMS completely. Hybrids are created by integrating OOP concepts into existing RDBMSs. This exploits RDBMS feats merged with OOP concepts (Satesh and Patel, 2009) via graph model. The model adopts the navigation system to provide fast search and movement across the network of objects using an object identifier as smart pointers to relate to the objects (Date, 1999; Codd, 1970). This hybrid will offer a more general data model by using objects to enhance RDBMS and incorporates relation not constrained by Codd's concept (which require that all data in database must be cast explicitly in terms of values in relations (Conrick, 2006). We model data to be represented via a digraph with trees on the nodes. Thus, the database seeks to extend RDBMS with non-relational features (Zhuge, 2008).

## E. DATA MINING TECHNOLOGY AND HEURISTICS

Data stored in databases possess valuable hidden knowledge, which can be engaged in fruitful decision making processes. It makes imperative the need to develop methods for extracting knowledge from such hidden data. Of the numerous methods proposed, data mining has been successfully used in this task. Data mining employs pattern recognition feats with statistical and mathematical techniques in discovering meaningful new correlations, patterns and trends by analyzing large amounts of data stored in the repositories (Seifert, 2004). Data mining has made impact in many applications to include management, web mining, marketing, customer relations, crime analysis, engineering, medicine, prediction, and mobile computing to mention a few (Chen et al, 2005; Li, Yang and Zhou, 2008). Data mining tasks can be classified into two: Descriptive- and Predictive-mining (Satesh and Patel, 2009).

Descriptive mining extracts vital characteristics and/or feats of data from databases. Examples are clustering, Association Rule Mining and Sequential mining; While Predictive mining derives hidden patterns and trends from data to make decisive predictions. Predictive mining techniques consist of a series of tasks such as classification, regression and deviation detection (Han et al, 2001; Coenen et al, 2004). An important tasks in data mining is classification, which aims to find a valuable set of models that are self-descriptive and distinguishable data classes or concepts, to predict set of classes with an unknown class label (Zhong, Fu and Zhou, 2006; Waiyamai et al, 2004). In transportation network, all highways with same structural and behavioral properties can be classified as a class highway (Shahrabil and Kainz, 1993). All forms of data mining, classification is used in applications such as credit approval, product marketing, and medical diagnosis (Kamber et al, 1997). So many techniques such as decision trees, neural networks, nearest neighbor methods and rough set-based methods enable the creation of classification models (Al-Hegami, 2007). Regardless of its potential effectiveness and that data mining appreciably enhances data analysis, the technology requires great effort is to be taken to integrate data mining technology with database system (Homan and Kovac, 2009).

## F. THE GRAPH SYSTEM

A graph-structure consists of a set of nodes or vertices that are connected together by a corresponding set of edges, links or relationships. It yields a system that has been adapted in many disciplines (such as mathematics, computing, sociology, biology, engineering etc) to represent relationship amongst a set of interactions nodes (actors or agents) in such a system. Nodes are represented as actor/agents; while its corresponding relationships are represented via edges or linked arcs – so that the system in general, studies how these agents, their relations and interaction (represented via the formation of node clusters and/or isolation) creates an effect that ripples throughout the system (Ojugo et al, 2015).

The graph plays a powerful role as networks to bridge local features that are existent within these actors or agents – so that they can blossom into a global (graph) pattern that helps users explain how effects of classification via clusters and isolation. Each agent plays a significant structural role that helps shape the graph's evolution in time, and they adapt themselves to various dimensions (as need arises) via relationships that determines how data flows in a system (Ojugo et al, 2014).

Mathematically, a graph is an abstract representation of a set of object that are connected by links. These interconnected objects represent *vertices*; while the links that connects a pair of vertices are *edges* (Izquierdo and Hanneman, 2008) denoted as G = (V,E). Each node $x \in$ V with edges $m \in$ E. Its edges indicates the direction of data flow: which is grouped into undirected and directed. A direction can either be self-linked (loop), single- or multi-link. Each node has a set of neighbors to which it is either linked to, or is isolated from. The links can be weak, strong or isolated, in terms of relations status as measured through *dyads D* (West, 2001). These links or edges describe what relations exist between actors. Graph of *single* relation among its nodes is *simplex* graph; while graphs with multiple relations are *multiplex* graphs (which are analyzed using various graph techniques/tools). An undirected relation implies co-occurrence, co-presence or bonded-tie where actors are of same level (such as siblings relations), or a *directed* relation with a superior and a subordinate model where data originate from a source and reaches target actor via a directed arrow (such as parent/child, employer/employee). If directed edge is reciprocated, it implies that both nodes are source and target at any given time (represented by bi-directional arrow). Some edges may be weighted using techniques to indicate the cost or penalty of movement from one actor to another actor (Diestel, 2005).

## G. GRAPH MODELS AND TOPOLOGY

Modeling is an abstracting activity motivated by a particular need or goal that sees to help bring specific facets of an unruly domain into a space that can help us structure and manipulate them. Since there are no natural representations of the world the way it really is – we then employ purposeful selections, abstractions, and simplifications, some of which are more useful than others for satisfying a particular goal. A difference between graphDB and other modeling techniques, however, is the close affinity between the logical and physical models. RDBMS require developers to deviate from natural language representation of the domain: (a) cajoling our representation into a logical model, and (b) forcing it into a physical model. This transformation introduces semantic dissonance between our conceptualization of the world and database's instantiation of that model. But, GraphDB shrinks this gap considerably as its models naturally fits with the way we tend to abstract the salient details from a domain using circles and boxes, and then describe the connections between these things by joining them with arrows (Robinson, Webber and Eifren, 2013).

GraphDB technologies are 'whiteboard friendly' and typical whiteboard view of a problem *is* a graph – since what users sketch in their creative and analytical modes maps closely to the data model implemented inside the GraphDB. These models also reduce the impedance mismatch between analysis and implementation that has plagued RDBMS and OODBMS over time. An interesting feat that continues to draw developers to its use is in how actors in the GraphDB communicate, how actors relate, and the facts that they also clearly communicate the kinds of questions pertinent in the domain.

Graphs are basically divided into three (3) models namely (Ojugo et al, 12015b):

- ✓ *ERDOS-RENYL MODEL* describes a random graph with nodes *n* connected to each other randomly, selecting from edges. Its grows via probability distribution of nodes given by, following the Bernoulli Degree where k is average connectivity in G. If probability *P* is small, then G has many isolated components (subgraph or nodes). But, if as large, then almost all nodes are connected (Pavlopoulos et al, 2011).
- ✓ *WATTS-STROGATZ* model introduced in 1999, describes a small-world graph model that deviates from the classical concept of random networks. Each node is sequentially inserted to the graph and linked to an existing node based on a probability that is proportional to its current degree in hierarchy. Graph grows via power-law distribution with probability P of the nodes with degree *k* proportional to $P(k) = k^{-\gamma}$, $\gamma = 3$ (Watts and Strogatz, 1998; Pavlopoulos et al, 2011). The graph is characterized and influenced by: (a) small path length (α) that defines the average/shortest path between a node pair and determines probability of node connected given a number of common neighbors. This property controls extent of a graph being filled with sparsely or densely connected components so that as α nears infinity, the graph becomes more a random graph), and (b) *large clustering* coefficient (q) is the average fraction of pairs of neighbors of a node that is connected to another. This feat determines probability of an edge to reconnect to another node in the graph. If the value of q is small, it denotes high clustering coefficient and large average path length, which leads G to become a random graphs; Else, as q tends to 0, it becomes more of a small-world graph (Schnettler, 2009 and Ojugo et al, 2014).
- ✓ *SCALE-FREE GRAPHS* describes Barabasi and Albert model of 1998. This graph reveals data about its dynamics from an evolutionary point, and like the small-world topology, it deviates from the classical random networks based on two feats namely: *growth* and *preferential attachment*. Its core idea hinges on the fact that it considers a network as an evolving entity so that it models the dynamics of network growth. The simple BA model is well-known and described by Albert and Barabasi (2002): Given a positive integer *m* in an initial network $G_o$, network evolves based on these rules in discrete time-process as thus:
  - *GROWTH:* At each time *j*, a new node of degree *m* is added to the graph or network.
  - *PREFERENTIAL ATTACHMENT:* For a node *x* in the graph, the probability a new node connects is proportional to the degree of *x*. We express $G_j$ for the network at time j and P(x,y) for the probability that the new node added at time k is linked to x in $G_{j-1}$ as in Eq. 1:

Each node inserted is sequentially linked to existing one via probability proportional to the existing node's current degree, in hierarchical fashion. This model generates a graph whose degree of distribution, asymptotically tends to a power law so that a node *x* of degree k is proportional to $P(k) = k^{-\gamma}$ and $2 < \gamma < +\infty$ (Albert and Barabasi, 2002; Pastor-Satorras and Vespignani, 2002; Pavlopoulos et al, 2011). Its variants as seem to follow power law degree such that if $\gamma \leq 3$, it yields a small world graph. But, if $\gamma \geq 3$, it yields scale-free graph and evolves with distribution of $2 < \gamma < +\infty$ (Barabasi and Albert, 1999; Dorogovtsev, Mendes and Samukhin, 2000).

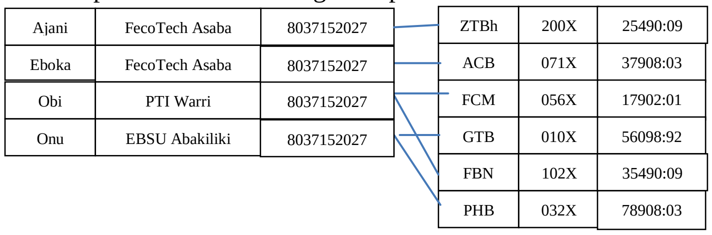
## H. THE GRAPH DATABASE (GRAPHDB) MODEL AND STRUCTURE

A GraphDB is an online database system with Create, Read, Update, and Delete (CRUD) techniques used to expose the graph model and generally used for online-transaction (OLTP) system. It is normally optimized for transactional performance and engineered with transaction integrity and operational availability in mind (Barmpis and Kolovos, 2014). Two feats to be explored in using GraphDBs technologies include:

- ✓ Storage: Most GraphDBs uses the native graph storage, optimized and designed for storing and managing graphs; while some serialize its data into a RDBMS, OODBMS or some other general-purpose data store.
- ✓ Processing: Some GraphDBs use index-free adjacency so that connected nodes physically point to each other in the database. Any database behaves like a graphDB (exposes a graph data model via CRUD operations) qualifies as graphDB database. This implies a significant performance advantages of index-free adjacency, and thus, use the term native graph processing to describe graph databases that leverage index-free adjacency.

*Figure 1: Many-to-Many relationship for 2-data structure*

The structure diagram is a schema representing the design of a network database consisting of boxes (which represents the record types) and lines (which represents the links). Owing from the model types as mentioned above, consider the E-R Graph consisting of two entity sets, client and account data, related via a binary, many-to-many relationship *depositor* with no descriptive attributes. Diagram specifies that a client can



have several accounts, and that an account can also belong to several different clients. Record type *client* is the entity set *client* with includes 3-fields: *client_name*, *client_address* and *client_phone*. Similarly, *account* is record type representing an entity set with 3-fields: *bank*, *account_number* and *balance* as:

```
type client = record
    client_name: string;
    client_address: string;
    client_phone: integer;
            end
type account = record
bank: string;
account_number: string;
balance: float;
end
```

| Name | Address | Phone |
|---|---|---|
| Ajani Emma | FecoTech, Asaba | 0803-715-2027 |
| Eboka Andrew | FecoTech, Asaba | 0802-345-1209 |

| | | |
|---|---|---|
| Obi Isioma | FecoTech Asaba | 0803-450-2313 |
| Onu Joseph | abakailki | 0803-420-9012 |

*Table 1: Client*

| Bank | Account No | Balance |
|---|---|---|
| ZTB | 2007176543 | 25490:09 |
| ACB | 0715290341 | 37908:23 |
| FCM | 0562304325 | 17902:01 |
| GTB | 0109982670 | 56098:92 |

*Table 2: Account*

The relation is replaced with link depositor is a a many-to-many relationship for which: (a) the relation *depositor* is a one-to-many from client-to-account such that *depositor* points from *account*-to-*client* record(s), and (b) relation *depositor* is one-to-one if link depositor has two arrows where one points from *client*-to-*account* record, and other points from *account*-to-*client* record type. Thus, for a many-to-many relation, the depositor relation will just show lines to link depositor to both record types (account and client) graphically represented as:

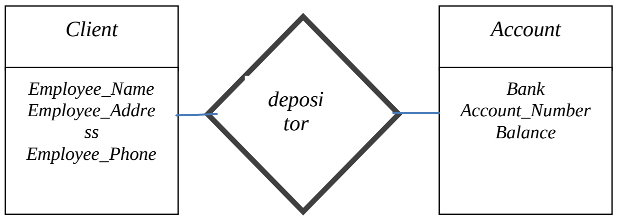


*Figure 1b: Sample Database diagrams*

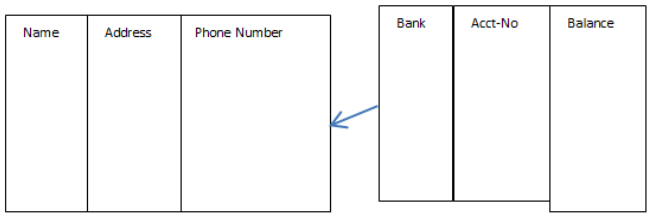


*Figure 2: a One-to-Many (1:M) Relation for the 2-data structure Diagram*

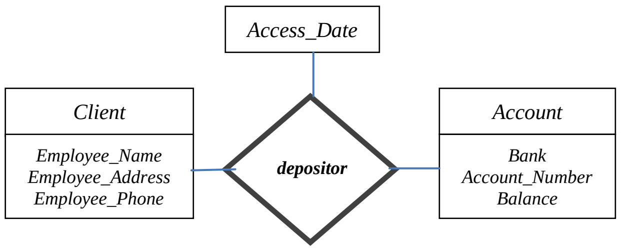


*Figure 3: a One-to-One (1:1) Relation for the 2-data structure Diagram*

The GraphDB schema described above can contain a number of client records linked to a number of account records as in fig 1b. With a many-to-many relationship, it shows that Obi and Onu both has 2-accounts each.

## I. PROPERTY OF GRAPHDB

A GraphDB has the following properties namely (Angles and Gutierrez, 2008; DeVirgillio et al, 2014):

- ✓ GraphDB initializes nodes as documents to store feats in the form of arbitrary key-value pairs. The keys are strings and the values are arbitrary data types.
- ✓ Relationships connect and structure its nodes with label and direction so that each GraphDB has start/end node with no dangling relationships. A relation's direction and label together, adds semantic clarity to structure of nodes.
- ✓ Relationships can also have properties – which is useful in providing additional metadata to a graph algorithms, adding additional semantics to relations (including quality and weight), and for constraining queries at runtime.

## J. RATIONALE FOR THE CHOICE OF GRAPH DB

- ✓ Modeling: Any task can be modeled as graph. And though the GraphDB provides a powerful, novel data modeling technique; It does not account that this in itself provide sufficient justification to replace a well-established, well-understood data platform. There must be an immediate and significant practical benefit.
- ✓ Motivation: GraphDBs is motivated with the existence of use cases and data patterns whose performance improves by one/more orders of magnitude when implemented, and whose latency is much lower compared to aggregates in batch processing. It offers performance benefit, that are extremely flexible and a mode of delivery that is aligned with today's agile software delivery practices as used in OOP and new paradigm of Agent-based programming.
- ✓ Performance: Its performance increase when dealing with connected data allows the integration with OODBMS, RDMS and NOSQL. It creates the needed hybrid that allow migration flexibility between these database models such that in cases where join-intensive query performance deteriorates as dataset gets bigger, the GraphDB tends to retain a relatively constant performance – because queries are localized to a portion of the graph. Thus, at run time, each query is proportional only to the size of the part of the graph traversed to satisfy that query, rather than the size of the overall graph.
- ✓ Flexibility: With developers and data architects, GraphDB seek to connect data as the application domain dictates (so that its schema and structure emerge in tandem with our growing understanding of the problem space as well as suits the real data and intricacies of the data) rather than being imposed upfront. GraphDB addresses this via its model and accommodates business needs in a way that enables IT to move at the speed of business. Graphs are naturally additive, allowing users to add new relations, new nodes and new subgraphs (components) to existing structure without disturbing existing queries and functions of application. Such feedback cum positive implications increase developer's productivity and reduces project risk. This flexibility implies that developers do not have to model the application in exhaustive detail ahead of time – which will consequently require a corresponding change as business requirements changes. The additive nature of graphs also means there are fewer migrations, which in turn reduces maintenance overhead and risk.
- ✓ Agility: Gives a developer the ability to evolve our data model in step in tandem with our OOP application, using a technology aligned to today's incremental and iterative software delivery practices. Modern GraphDB equips the developers with abilities of frictionless development and

graceful systems maintenance. In particular, the schema-free nature of GraphDB coupled with its API (application programming interface) and query – empowers users to evolve in a controlled manner. Also the GraphDB lacks schema-oriented data governance mechanisms in RDBMs and other data models as it calls for a more visible and actionable kind of governance. The model queries assert biz rules that depend upon the graph, which aligns well with today's agile and test-driven software development practices, allowing graph database–backed applications to evolve in step with changing business environments.

## III. PROPOSED GRAPH FRAMEWORK

### A. MODELING THE DATABASE

RDBMS and OOPs have different programming paradigms that are not very compatible. Many OOPs do not have any standard method for accessing data type information at run time. Thus, many developers seek means to parlay these feats via their application's interface design, which is specific of the database model adapted. Our hybrid is achieved and modeled via one-of-this strategies: (Satesh and Patel, 2009; Robie and Bartels, 2013):

- ✓ Model the Database in your Application Classes – A user can build an OO-application around a relational model, where each object must be instructed on how to retrieve and store data from the database. This strategy requires extra programming in every class, and does not result in artistic clean program design (for details of the database as implemented must reflect in every class) and structure of data for the OOP is not closely related to the tables of relational databases. Thus, this often renders the infused code, non-trivial.
- ✓ Model Application in Database – Here, a user reflects the object model in a RDMS. This strategy is much harder the previous one because RDBs have limited data types, and it requires significant coding in OOP application as many aspects of objects cannot be expressed directly in RDB (e.g. inheritance, pointer, polymorphism, collections).
- ✓ Row-Orientation Interface – In case the application only needs simple data-types with not have many relationships – user can limit his/her interfaces to row-oriented access. This greatly simplifies development. In many databases, it is useful to define views that present data to the program in a convenient way. Some applications need to store complex data; But, it then exchanges only simple data with RDB. In such cases, it is imperative and wise to use an OODBMS for its data and use simple tables and views to share data with RDB.

### B. THE EXPERIMENTAL GRAPHDB

Suppose our GraphDB as above includes a relationship with descriptive attributes – its E-R diagram transformation is more complicated. A link cannot contain any data value, so a new record type is created to link what needs to be established. If, we derive the attribute *access date* to *depositor* relationship (to indicate the most recent time a client accessed an account) – then, our new derived E-R diagram transforms to:

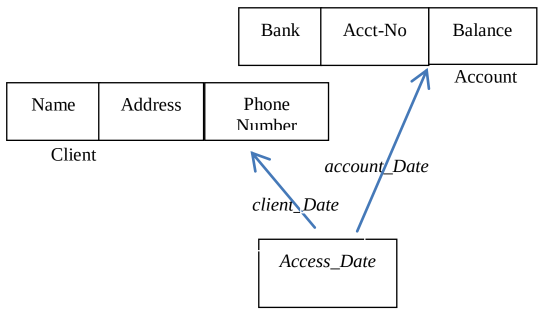


*Figure 4: Adding the Access_Date feat to E-R Diagram*

Eventually, if we add some other attributes that seeks to find data about a particular user whose account is domiciled in one branc and can also be accessed from other parts of the State and/or Country – then we implore data mining techniques with queries that uses the GraphDB to access the record in want.

## IV. DISCUSSION AND FINDINGS

The GraphDB methodology allows users to develop a database system that acts as a backend for most applications, which perform consistently better in its data manipulation and queries than other models and hybrids. The model drastically reduces the space overhead, the time complexity of the overall process and its search/manipulation of data with respect to any other competitor strategy that spends much time traversing a large number of edges. Variants (version) of the GraphDB have been successfully implemented by other studies such as Angles and Gutierrez (2008), De Virgilio et al, (2013). These studies also agreed that significant result was shown in the speed-up between some strategies.

The concept of storing and managing graph-data natively is quite old and has been recently re-born with the advent of the Semantic Web and other emerging application domains, such as social networks and bioinformatics. This new interest has led to the development of a number of GraphDBs that are now becoming quite popular. In spite of this, current approaches rely on best practices and guidelines based on typical design patterns, published by practitioners in blogs or only suited for specific systems (a design pattern based on join operations that need to be performed over the database). But, assumptions on the way in which the database under development is accessed and many other approach relies only on the knowledge of conceptual constraints that can be defined with the ER model as mentioned in Section II. Also, GraphDB provides a system-independent intermediate representation that makes it suitable for any application to integrate. The many feats of GraphDB can allow effective and efficient migration of data from a relational to GraphDB. In this work, we consider a different scenario where the database needs to be built from scratch.

## V. CONCLUSION

Various mechanisms are available in the GraphDB model for updating records in the database, which allow users to create and delete records (via the store and erase operations), as well as the modification (via the modify operation) of the content of existing records. The use of *connect*, *disconnect* and reconnect operations helps to insert or remove records from a particular set occurrence. So, when new set is defined, the user must specify how member records are to be inserted, and under what conditions it is moved from one set occurrence to another. Thus, a newly created member record can be added to a set occurrence either explicitly or implicitly. This distinction is specified at set-definition time via the insertion is statement with the manual and automatic insert-mode options. There are various restrictions on how and when a member record can be removed from a set occurrence into which it has been inserted previously. These restrictions are specified at set-definition time via the retention is statement with the fixed, mandatory, and an optional retention options. Implementation techniques for the GraphDB model exploit the restrictions of the model to allow the physical representation of GraphDB sets without the need for variable-length records. A GraphDB set is represented by one ring structure for each occurrence.

## REFERENCES

[1] Alam, M and Wasam, S.K., (2006). Migration from Relational Database into Object Oriented Database, Journal of Computer Science, 2(10), pp. 781-784, 2006.
[2] Albert, R and Barabasi, A., (2002). The statistical mechanics of complex networks, Reviews of Modern Physics, 74, pp47-97.
[3] Al-Hegami, A.S., (2007). Classical and Incremental Classification in Data Mining Process, International Journal of Computer Science and Network Security, 7(2).
[4] Angles, R and Gutierrez, C.,(2008). Survey of graph database models, ACM Computing Surveys, 40(1), pp 1 – 12
[5] Bagui, S., (2003). Achievements and Weaknesses of Object-Oriented Databases, J. of Object Technology, 2(4), pp. 29-41.
[6] Barabasi, A.L and Albert, R., (1999). Emergence of scaling in random networks, Science, 286, pp509-512
[7] Barmpis, K and Kolovos, D.S., (2014). Evaluation of Contemporary Graph Databases for efficient Persistence of Large-Scale Models, Journal of Object Technology, Association Internationale pour les Technologies Objets JOT 2014, Online at http://www.jot.fm.
[8] Bloom, T and Zdonik, S., (1987). Issues in the design of object-oriented database programming languages, In Proceeding of ACM SIGPlan Notices, December 1987, [online]: https://www.researchgate.net/publication/221321149
[9] Codd, E.F., (1970). A relational model of data for large shared data banks, Communications of the ACM, l(13), pp.377-387.
[10] Coenen, F., Leng, P and Goulbourne. G., (2004). Tree Structures for Mining Association Rules, Journal of Data Mining and Knowledge Discovery, 15, pp. 391-398.
[11] Conrick, M., (2006). Health informatics: transforming healthcare with technology", Thomson, ISBN 0-17-012731-1.
[12] Date, C., (1999). When an extension is not an extension? Intelligent Enterprise, 22(8), pp 190-201
[13] De Virgillio, R., Maccioni, A and Torlone, R., (2014). Model-driven design of graph databases, in E. Yu et al. (eds.): ER 2014, pp. 172–185, 2014, Springer International publishing Switzerland 2014
[14] Diestel, R., (2005). Graph theory (3ed), Graduate text in Mathematics, 173, Springer-Verlag, ISBN: 3-540-26182-6.
[15] Dorogostev, S., Mendes, J and Samukhin, A., (2000). Structures of growing networks with preferential linking, Physical Review Letters, 85(21), pp4633-4636.
[16] Fong, H.J., (1997). Relational databases with object oriented designs and implementation for modern applications, Databases, 51(2), pp133 – 145.
[17] Han, J and Kamber, M., (2001). Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers.
[18] Homan, J.V and Kovac, P.J., (2009). A comparison of relational database model and the associative database model, Issues in Information System, 10(1), pp 208 – 213
[19] Leavitt, N., (2000). Whatever Happened to Object-Oriented Databases?, IEEE Computer Society, 33(8), pp. 16-19
[20] Li, L., Yang, B and Zhou, F., (2008). A framework for Object-Oriented Data Mining, Proceedings of 5th International Conference on Fuzzy Systems and Knowledge Discovery, 2, pp: 60-64.
[21] Izquierdo, L.R and Hanneman, R.A., (2006). Introduction to formal analysis and social networks using mathematica, {online}: http://faculty.ucr.edu/.../mathematica_networks.pdf, last retrieved December 3, 2014.
[22] Kamber, M., Winstone, L., Gong, W., Cheng, S and Han, J., (1997). Generalization and Decision Tree Induction: Efficient Classification in Data Mining, Proceedings of International Workshop Research Issues on Data Engineering, pp. 111-120, 7-8 April, Birmingham, UK.
[23] Nunn-Clark, K., Hunt, L., Hooi, T and Gnanasekaraiyer, B., (2003). Problems of Storing Advanced Data Abstraction in Databases, In Proceedings of the First Australian Undergraduate Students' Computing Conference, pp. 59-64, 2003.
[24] Ojugo, A.A., Ben-Iwhiwhu, E., Kekeje, D.O., Yerokun, M.O and Iyawa, I.J.B, (2014a). Malware propagation on time varying graphs, International Journal of Modern Education and Computer Science, 4(9), p24-37.
[25] Ojugo, A.A., Ben-Iwhiwhu, E., Yoro, R.E and Chiemeke, S.C., (2014b). Decision diffusion predictor model for socialgraphs, http://www.researchgate.net/publications/261287430_Decision_predictor_model_for_social_graphs.pdf,last retrieved July 2016.
[26] Okonta, E.O., A.A. Ojugo., U. Wemembu and D. Ajani., (2015). Embedding quality functional deployment in

software engineering development, West African Journal of Industrial and Academic Research, 10(1): 50-64.

[27] Pastor-Satorras, R and Vespignani, A., (2002). Epidemics and immunization in scale-free networks. Handbook of Graphs and Networks: From the Genome to the Internet.

[28] Pavlopoulos, G.A., Secrier, M., Moschopoulos, C.N., Soldatos, T.G., Kossida, S., Aerts, J., Schneider, R and Bagos, P., (2011) Using graph throy to analyze biological networks, BioData Mining, 4(10), http://www.biodatamining.org/contents/4/1/10.

[29] Robie, J and Bartels, D., "A comparison between relational and object oriented databases for object oriented application development, White paper for Poet Software corporation, 800 - 950- 8845

[30] Robinson, I., Webber, J and Eifrem, E., (2013). Graph databases, O'Reilly publications, ISBN: 978-1-449-35626-2.

[31] Satesh, A and Patel, R., (2009). Use of object-oriented concepts in databases for effective mining, International Journal on Computer Science and Engineering, 1(3), pp206-216

[32] Schnettler, S., (2009). A small world on feet of clay? A comparison of empirical small-world studies against best-practice criteria, Social Networks, 31(3), p179-189, doi:10.1016/j.socnet.2008.12.005.

[33] Seifert, J.W., (2004). Data Mining: An Overview, CRS Report for Congress.

[34] Shahrabil, B.A and Kainz, W., (1993). Implementation Approach for Object-oriented Topographic Databases using Standard Tools," In Proceedings of Eleventh International Symposium on Computer-Assisted Cartography, pp. 103-112, 30 October-1 November, Tehran, Iran.

[35] Waiyamai, K., Songsiri, C and Rakthanmanon, T., (2004). Object-Oriented Database Mining: Use of Object Oriented Concepts for Improving Data Classification Technique, Lecture Notes in Computer Science, 3036: pp 303-309.

[36] Watt, A and Eng, N., (2016). Database design – (2 Ed.), [online]: http://open.bccampus.ca, last accessed June 23 2016.

[37] Watts, D.J and Strogatz, S.H., (1998). Collective dynamics of small world networks, Nature, 393, pp440-442

[38] West, D.B., (2001). Introduction to graph theory (2ed), Upper Saddle River, Prentice Hall, ISBN: 0-13-014400-2

[39] Zhong, J., Fu, Y and Zhou, J.L., (2006). A Classification Approach Based on Evolutionary Neural Networks, International Journal of Computational Intelligence Research, 2(1), pp. 72-75

[40] Zhuge, H., (2008). The Web Resource Space Model, Web Information Systems Engineering and Internet Technologies Book Series 44. Springer. ISBN 978-0-387-72771-4, 2008