Graph Databases

Adrian Silvescu, Doina Caragea, Anna Atramentov
Artificial Intelligence Research Laboratory
Department of Computer Science
Iowa State University
Ames, Iowa 50011

Abstract

Gathering huge amounts of complex information (data and knowledge) is very common nowadays. This calls for the necessity to represent, store and manipulate complex information (e.g. detect correlations and patterns, discover explanations, construct predictive models etc.). Furthermore, being autonomously maintained, data can change in time or even change its base structure, making it difficult for representation systems to accommodate these changes.

Current representation and storage systems are not very flexible in dealing with big changes and also they are not concerned with the ability of performing complex data manipulations of the sort mentioned above. On the other hand, data manipulation systems cannot easily work with structural or relational data, but just with flat data representations. We want to bridge the gap between the two, by introducing a new type of database structure, called Graph Databases (GDB), based on a natural graph representation. Our Graph Databases are able to represent as graphs any kind of information, naturally accommodate changes in data, and they also make easier for Machine Learning methods to use the stored information.

We are mainly concerned with the design and implementation of GDB in this paper. Thus, we define the Data Definition Language (DDL) that contains extensional definitions as well as intentional definitions, and Data Manipulation Language (DML) that we use to pose queries. Then we show conceptually how to do updates, how to accommodate changes, and how to define new concepts or ask higher order queries that are not easily answer by SQL language. We also show how to transform a relational database into a GDB.

Although, we show how all these things can be done using our graph databases, we do not implement all of them. This is a very laborious project that goes much beyond our class project goals. We do implement the graph databases, show how we can ask queries on them, and also how to transform relational databases into graph databases in order to be able to reuse relational data already existent.

1. Introduction

Huge amounts of information (data and knowledge) become increasingly common in nowadays sciences. The sheer volume and diversity make it increasingly difficult to the unaided contemporary scientist to be able to exploit this information at its full potential. This observation calls for automatic and semi-automatic (human-driven, but not in every single detail) methods of acquiring, integrating, representing and manipulating complex data (detecting correlations and patterns, discovering hidden variables, explanations, a-priori unknown regularities, constructing predictive models etc.).

Usually data resides in multiple locations, is autonomously maintained, and may change in time, thus posing even more challenges. Being autonomous, it is possible that the data sources don't have a unifying schema or we cannot control their schemas. Furthermore, these schemas can be dynamically modified, and it may be difficult to accommodate changes. In this setting, several major issues arise: how to structurally match schemas of different data sources, how to obtain a global schema from multiple schemas, and how to change the global schema according to possible changes in data sources schemas.

Sometimes the attempts to solve these problems depart from the traditional approach to databases, namely RDBMS, leading into the realm of Object-Oriented and XML technologies. The new approaches can deal with increasingly complex data, namely objects in Object-Oriented Databases and trees in XML. However, neither the objects nor the trees can naturally represent graphs, which are the most general data structure. Subsequently, they do not support natural queries on graphs. Moreover, when changes in schema of the data sources occur, these approaches may lead to major restructuring (redesigning) of global schema. This may imply very sophisticated and expensive operations, such as renormalization, reindexing etc., which may not be performed automatically. Thus, the necessity to represent, store and manipulate complex data makes RDBMS somewhat obsolete.

We identify three major problems that can arise in the kind of dynamic, distributed, autonomous environments described above. First, *violations of the 1NF* can appear when dealing with multi-valued attributes, complex attributes, or combination of multi-valued and complex attributes. We denote this problem by [**P1**] in what follows. Second, when acquiring data from autonomous dynamic sources (e.g. gene expression data) or from Web (e.g. data about restaurants), a good data representation should be able to accommodate changes not only in data, but also in the data schema. We call this problem [**P2**]. It should be noted that RDBMS might require schema renormalization in such cases, which is not always easy to do. The third problem, which we denote by [**P3**], refers to the ability of a database to *represent data in a unified way*. Thus, it would be very good if data, knowledge (e.g. schemas), queries (or more generally, goals) and models (e.g. concepts) would have a unified representation. This would give us a better understanding of a data problem.

We should note that the three problems mentioned above are very relevant to the Scientific Discovery (SD) process using Artificial Intelligence (AI), and especially Machine Learning tools. This is because, much of the data available for SD is structural in nature, needs be often updated, can change easily in structure etc. Although most of the traditional machines learning algorithms work with data found in flat data representations, currently there is a considerable tendency to design intelligent learning algorithms that can deal with structural [Cook and Holder, 2000] or relational data [Getoor et. al., 2001]. However, these approaches, one based on graphs structures, the other on relational databases, do not take into consideration the problems [**P1**], [**P2**], [**P3**] described above. They work with data that does not change in time, and they are not dealing with any updates. Also, they do not address the unified representation problem. The problem they solve is to extract useful knowledge from data that is already represented as graphs or as relations. In [Getoor et. al., 2001], data and query do not have a unified representation at all. Although, in [Cook and Holder, 2000] the authors regard both data and query as graphs, they are not able to put together data and knowledge. Furthermore, they are not interested in these structures from a databases perspective at all, but just from a learning perspective. However, we want to stress on the fact that more and more graphical representations are used in ML lately [Jordan, 1998; Pearl, 2000; Jensen, 2001]. Although quite distant from what we are doing here, this suggests the importance of being able to deal with the natural graph representation of information.

We aim at designing database structures that can take into account problems [**P1**], [**P2**], [**P3**], facilitate the application of learning and reasoning methods on these structures, and also be able to reuse data that is already stored in relational databases. We call our newly designed structures *graph databases,* and we think about them as being at the core of information storage, information integration, and information exploration. The big picture showing how all these components come into place into the DS process is shown in Figure 1.

We are not interested in the integration process in this paper. We can assume that the available information is already integrated [Reinoso et al., 2001]. The problem that we are trying to solve is how to represent this integrated available information in a way that makes it easier to accommodate changes, updates, and also provides a unified representaion for data and knowledge, so that the learning process is facilitated.

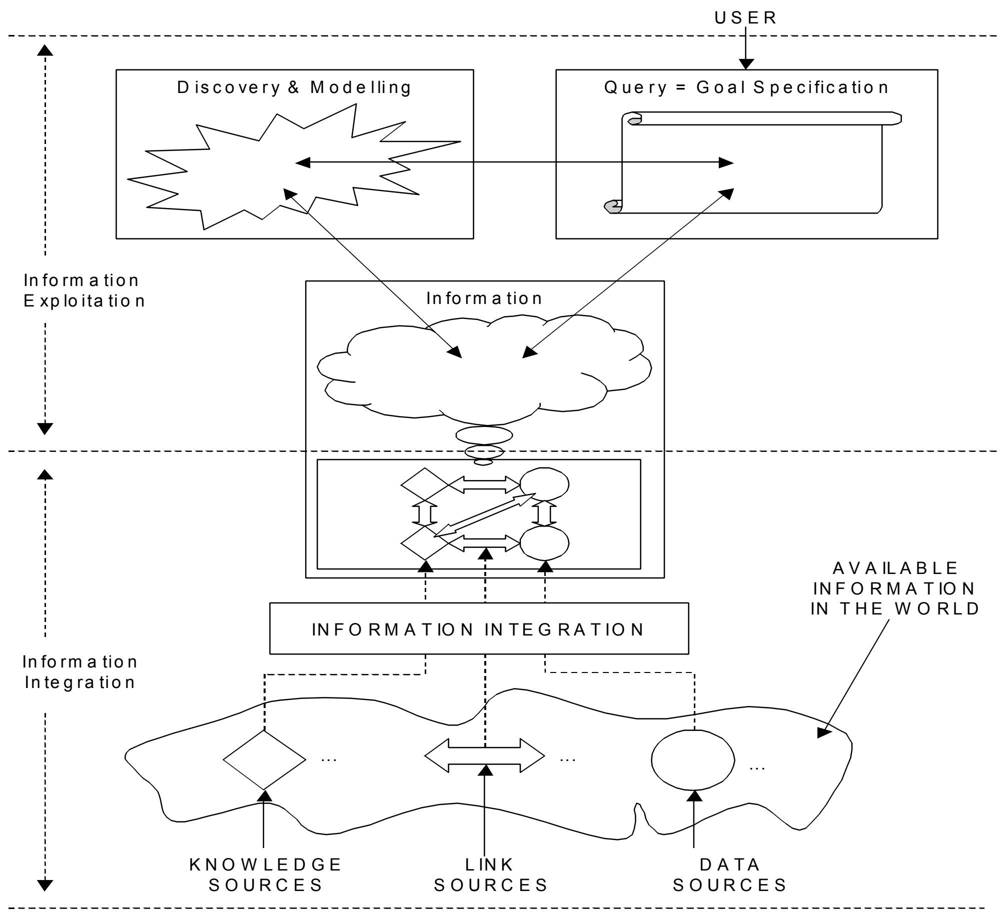


Figure 1: Big Picture of the Discovery Science Process

2. Related Work

We mentioned above that RDBMS systems are somewhat obsolete regarding the ability to represent, store and manipulate complex information that can change in time. Thus, they might require very sophisticated and expensive operations, such as renormalization, reindexing etc., which may not be performed automatically. Schema renormalization in such cases is nor desirable neither easy to do.

Therefore, the attempts to solve these problems depart from the traditional approach to databases, namely RDBMS, leading into the realm of Object-Oriented and XML technologies. The new approaches can deal with increasingly complex data, namely objects in Object-Oriented Databases and trees in XML. However, neither the objects nor the trees can naturally represent graphs, which are the most general data structure. Subsequently, they do not support natural queries on graphs. Moreover, when changes in schema of the data sources occur, these approaches may lead to major restructuring (redesigning) of global schema.

We briefly mention some of the existing older and newer databases systems, showing how they address the problems [**P1**], [**P2**] and [**P3**] that we are interested in. We do not go into too many details, because, as we mentioned, we are not only interested into database system appropriate for representing, storing and querying data, but also in systems that facilitate the information extraction process. On the other hand, we do not talk more about graphical learning methods [Jordan, 1998; Pearl, 2000; Jensen, 2001; Cook and Holder, 2000; Getoor et. al., 2001] because they assume fixed representations, not appropriate for changes, being concerned only with the learning part. They will be relevant to a discussion about how to use GDB for learning, which is beyond the purpose of this current paper, but very probable in future work.

To be more precise, we are not aware of any system that tries to bridge the gap between representation and learning.

In terms of existing newer and older databases approaches, more or less related to our approach, it worth mentioning:

- OO Databases that address problems [**P1**], [**P2**]. They also have a graph representation, but their main criticism is that they are procedural, not declarative, making the querying process more laborious.
- XML Databases address problem [**P1**] (and somewhat [**P3**]), but they assume a more restricted tree representaion.
- OORDBMS address problem [**P1**] using graphs with foreign keys.

Others, not very successful approaches:

- Datalog, which is a more efficient, crippled version of Prolog (gives up some benefits for some efficiency).
- Network Models that are based on graphs and stay at the core of OOD, so they are also procedural, not declarative.
- Hierarchical Models that are based on trees and stay at the core of XML technology.

Given that fact that none of these existing approaches addressed reasonable all the problems we are interested in, we have a strong motivation to design such a system. Opting for the graph structures was the effect of the current tendency toward graphical data and model representations, and also the effect of the book knowledge representation by Sowa [Sowa, 2000], which shows that any information can be easily represented by conceptual graphs.

3. Proposed Work - GDB

We propose a new kind of databases called Graph Databases (GDB) as a solution to the problems [**P1**], [**P2**] and [**P3**] described in the introduction. Our graph databases have a general graph representaion of data as opposed to relational databases. Thus, we develop the equivalent of a database system for graphs. Assuming that the integration problem is solved in advance [Reinoso et al., 2001], our graph databases are appropriate for acquiring data from different sources (maybe in different formats) by presenting them as graphs to our graph databases. Our system is able to accommodate changes in data structure, and easily perform updates. Also, all data, knowledge, concepts, models etc. are regarded in a unified way, having the same graph structure.

For our purpose, the proposed system can be seen as a generalization of databases. This is because every database (schema + data) can be expressed as a graph [Sowa, 2000]. Even more, our system being much more expressive, it will be able to integrate a bigger variety of data sources (e.g., hypertext). Also, it will be able to answer more complex queries, some of which cannot be easily expressed in SQL (e.g., “show me all the tables that contain name John”).

In order to cope with any possible change in the structure of the data, an ideal schema should be "absolutely normalized". Such an absolutely normalized schema would be one in which all the entities would have only one column, and all relationships would have only two columns. This is basically a graph. This would imply that all the changes in a schema would affect the graph only locally, therefore not leading to the restructuring of the whole "schema" (graph). This is how we will achieve dynamicity in our system.

We are also addressing the main criticism of network models, which use legacy and procedural methods for accessing data, by defining declarative methods of expressing queries over graphs. It may seem that we are trying to resurrect a dead old model (network model). But from a certain point of view, the same is happening with XML proponents who are presumably trying to resurrect the hierarchical model [Silberschatze et al., 1996], so it may not be a bad strategy after all. The main difference between network models and our graph models is that every attribute in our model is going to be represented as an entity set.

Note that absolute normalization, in the same way as Boyce Code Normal Form, is going to lead even to a bigger extent to the loss of dependency preservation and decreasing query performance due to additional joins that need to be performed. However this is a trade-off that we need to make in order to preserve flexibility. Also with respect to dependency preservation, we don't think this is a big loss since we are interesting mainly in quering. The decreasing query performance can potentially be fixed using techniques similar to optimization techniques in databases applied to graphs.

In order to define the GDB we need to specify: the *Data Definition Language* (DDL), the *Query Language* (more generally, Data Manipulation Language - DML) and the Informal Semantics of the DDL and DML languages. We will also show how to convert existing relational databases (RDBMS) into the GDB DDL to facilitate the transition to GDBs. We discussed briefly the difference between databases having unique instance identifiers and databases having foreign keys. We needed to make a design choice for our databases regarding this issue and we chose to have *unique instance identifiers*, as opposed to foreign keys. This will allow us to cope easier with changes. Thus, our approach is closed in spirit to the Object Oriented Databases, but the declarativity is not an issue for GDB, as we will show, although it is an issue for OOD.

3.1. Data Definition Language

When designing a graph databases, we need to start the definition by showing how to represent a graph. The basic building block of a graph used in our paper is showed in Figure 2.

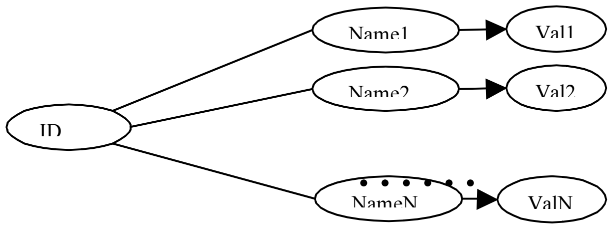


Figure 2: Graph Building Block

The formal definition of this graph can be written as: ID:(Name1=Val1,…,NameN=ValN). Putting together a lot of these graphs, we can represent as a big graph most of the real world problems.

There are three kinds of definitions that we need to specify in order to completely define the DDL. First, we need to be able to represent *facts* (data instances) that are graphs representing *extensional definitions*. In our DDL, we denote facts by a graph followed by dot (e.g. *G1*.). Second, we incorporate into our databases *intentional definitions*, for which we do not explicitly store the data, but just a way that makes easy to retrieve that data on demand. An intentional definition is usually composed from two parts, a head (a graph *G1*) and a tail (another graph *G2*) linked together by the definition sign :- and followed by dot (.). For example, *G1:-G2*., denotes an intentional definition. Third, our GDB contains also *procedural definitions*, which prescribe a way of finding a graph *G1* based on some procedural function. For example, this kind of definitions can be used to define aggregate operators from SQL language. A procedural definition looks like*: G1 :- PROC f(x1,…,xn).,* where *f* is the function that needs to be computed, and *x1,…,xn* are its actual parameters.

3.2. Example - from RDBMS to GDB

In order to better explain our proposed data structure, we show how a relational database can be transformed into a graph databases using a very well known example of RDBMS. We assume that we have three tables, Sailors, Boats and Reserves, whose schemas are given below:

- Sailors(sid:integer, sname:char(10), rating: integer, age:real)
- Boats(bid:integer, bname:char(10), color:char(10))
- Reserve(sid:integer, bid:integer, day:date)

The data available in the database is shown in the following tables:

Sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Reserves

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Boats

bid	bname	color
101	Interlake	red
102	Clipper	green
103	Marine	red

Some part of the corresponding part is shown in Figure 3.

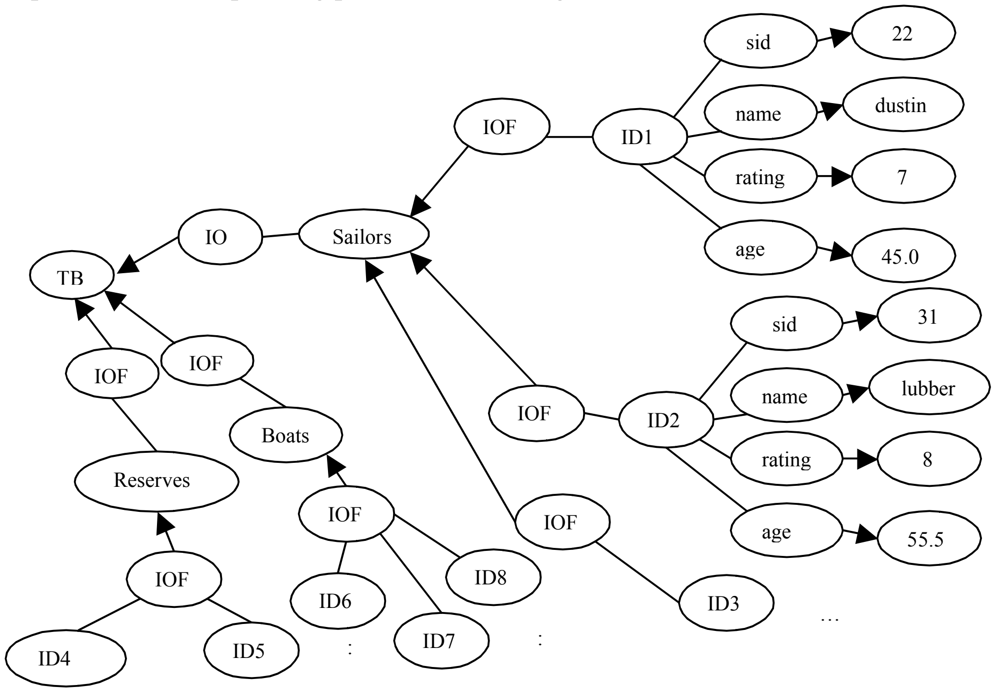


Figure 3: Partial GDB for Sailors, Reserves, and Boats

The graph above can be described formally using several extensional definitions such as:

ID1:(sid=22, name="Dustin", rating=7, age=45.0).
ID6:(bid=101, bname="Interlake", color="red").

We show how to deal with deal with foreign keys in Figure 4. Here we construct edges corresponding to foreign keys between IDs from different related tables, and thus the original edges representing the foreign key attributes in the small subgraphs are deleted.

The formal definition for a subgraph involving foreign keys is as follows:

ID4:(sailor=ID1, day="10/10/96", boat=ID6).

The storage necessary for the GDB is at most three times bigger than the storage necessary for the corresponding relational database, as we store at most two more nodes for each instance value in a database. However, GDB come with some advantages in terms of dealing with the problems we are interested in.

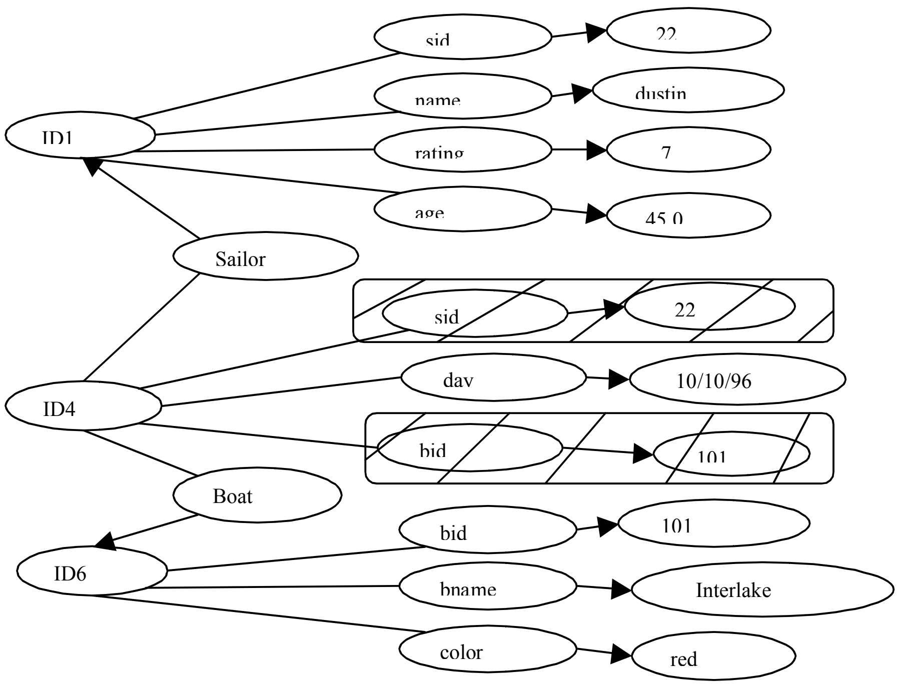


Figure 4: Foreign Keys Representation

3.3. Defining New Concepts in GDB

In Figure 5, we show how to add a new concept, called Granson, to a database containing the concepts Son and Person. Semantically this definition is equivalent to adding an edge called GrSon between _ID1 and _ID2 to the graph on the right (we denote by *_name* a variable, a name that is uninstantiated). However, this edge is not really added, but can be produced on demand based on the given definition.

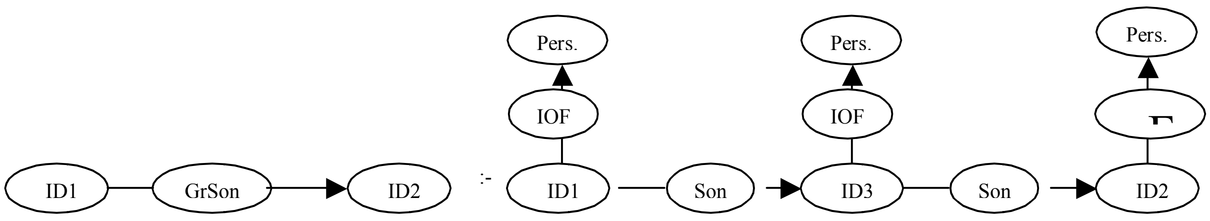


Figure 5: Grandson Concept Definition

Formally the concept described in Figure 5 can be written as: ID1:(GrSon=_ID2) :- _ID1:(IOF="Person",Son=_ID3), _ID3:(IOF="Person", Son=_ID2), _ID2:(IOF="Person").

3.4. Query Language

We represent a query by a graph similar to the graph representing an intentional definition, and we call it *query graph* (*CG*). Actually, queries are graphs to be matched in the big graph containing facts, intentional and extensional definitions. A query can be written formally as: *Query :- QG.*

In Figure 6, we show how to represent the query: "find the names of all sailors who have reserved a red boat".

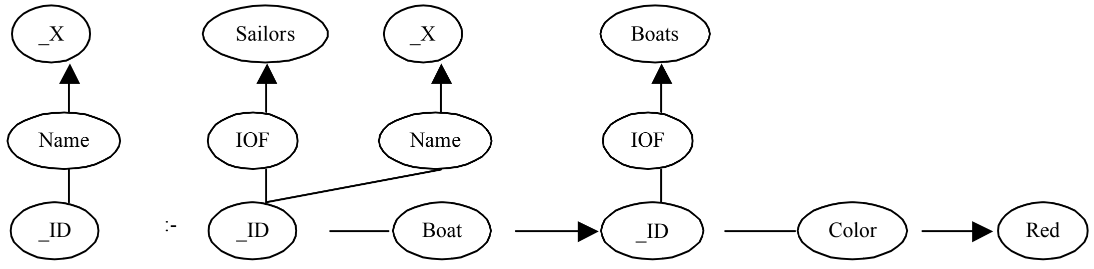


Figure 6: Graph Query Representation

The semantic definition of asking a query can be informally visualized as in Figure 7. Thus, when a query is asked, it is matched with the big data and knowledge graph. The answer can lie in facts or in the extended graph, but sometimes even in the intersection of the previous two.

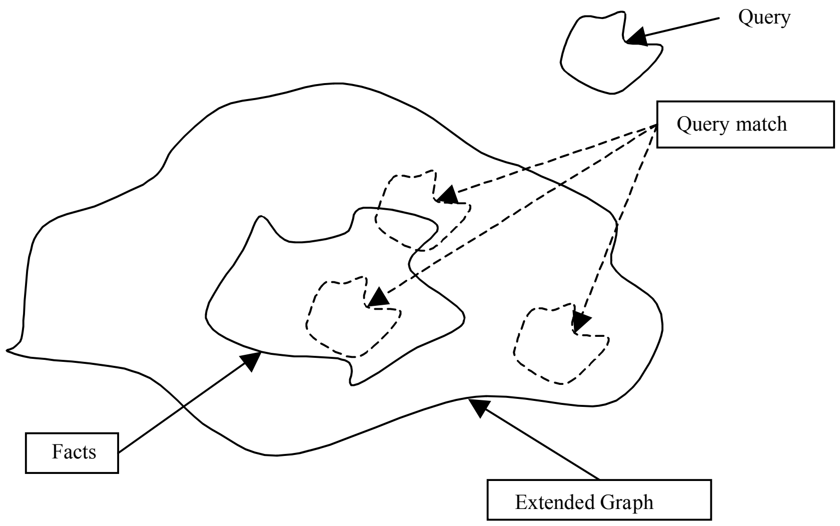


Figure 7: Informal Semantic for Query Match

3.5. Updates

Another feature that our system offers is the ability to handle updates very easily. In general, the formal description of an update operation can be written as: *MODIFY (QryGraph, UpdateList)*.
For example, suppose that we want to update our Grandson concept according to the additional constraint: Add to all GrandSons the money of the Grandparent as a potential inheritance. The update rule can be written as:

MODIFY (_ID : (GrSon = _ID1), (=> NEWID:(IOF = POT_INHER, BENFICIARY = _ID1, AMOUNT = _AMNT: [_ID:(Money = _AMNT)]))).

3.6. Changes

Besides updates, we can easily incorporate changes, simple (Figure 8) or more complicated (Figure 9) using our GDB. Thus, in Figure 8 records about books are shown. If initially just the first author was considered, subsequently we may want to be able to query also about the second author. So each book instance will have two authors instead of just one. Our GDB can be easily changes to accommodate this requirement, as opposed to ORACLE databases where the whole schema need to be changed by the DB manager.

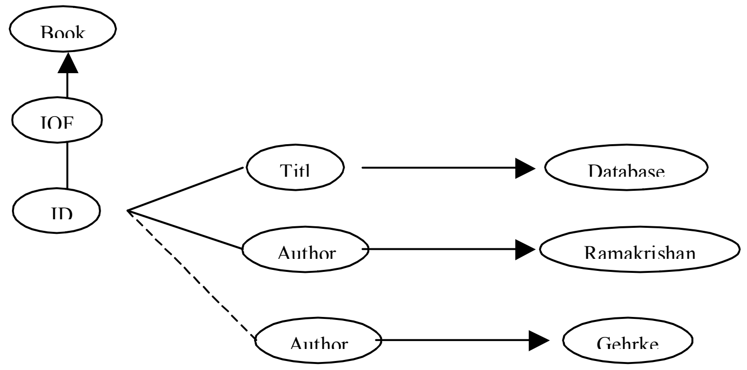


Figure 8: Simple Change

In Figure 9, we show some gene expression data. For each instance, we record the value of expression. However, at some point the data providers, realize that this value is a function of some variables and they decide to provide the values of those variables instead of the expression value. We want to record these values, but also to be able to compute on the fly the expression value. Figure 9 shows how we can easily do this in our GDB.

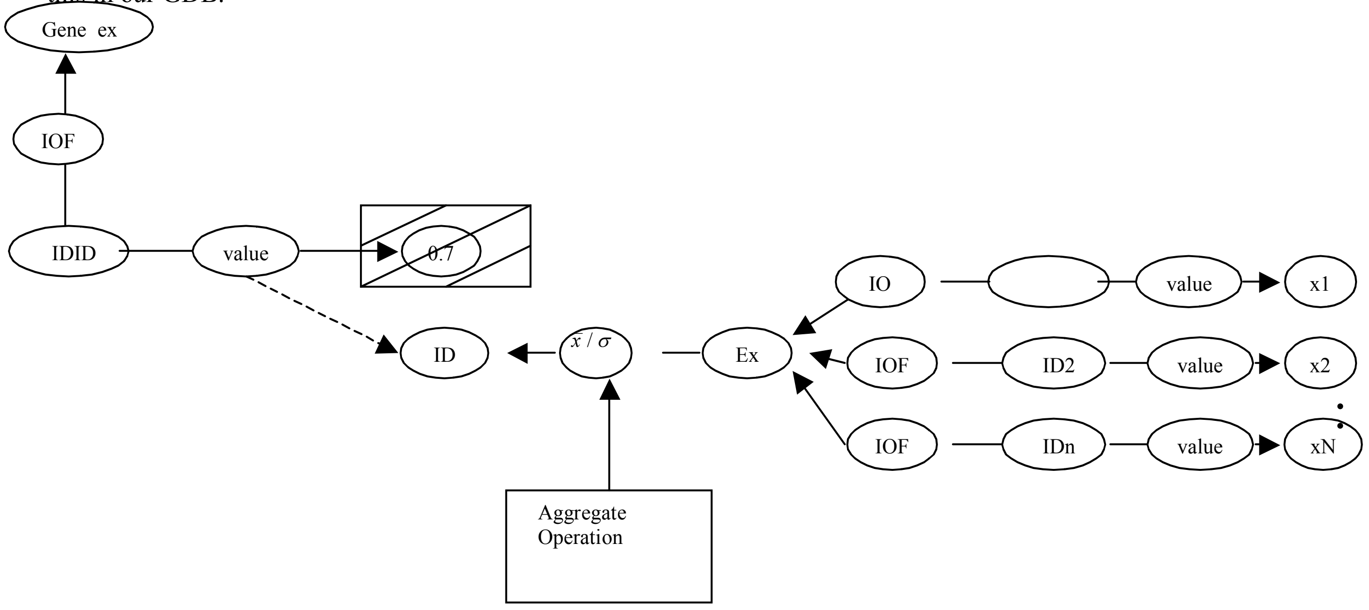


Figure 8: Complex Change

3.7. High Order Queries

As opposed to SQL language, GDB query language allows us to ask high order queries. For example, we can ask for all the fields from Tables that contain the name John. This can be easily written as:

ID:(Name=_X) :- _ID1:(IOF=Tables), _ID2:(IOF=_ID1, _X="John").

4. Performance Evaluation

As shown before, we designed a new kind of database, based on graphs that can take into account important problems that appear when dealing with autonomous dynamic databases. To make some preliminary steps toward performance evaluation of our system, we implemented our GDB data definition language and data query language. Our program inputs two files, one containing data and knowledge (facts and definitions, as shown below), the other contains the query, also in the definition format. The output file contains the answer to the query in formal graph representaion.

```
AIRPLANE:(InstanceOf = Tables).
AIRPLANETYPE:(InstanceOf = Tables).
AIRPORT:(InstanceOf = Tables).
CANLAND:(InstanceOf = Tables).

AIRPLANE_0:(InstanceOf = AIRPLANE, AIRPLANEID = B1, TOTALNOOFSEATS = 279, TYPENAME = AIRPLANETYPE_0).
AIRPLANE_1:(InstanceOf = AIRPLANE, AIRPLANEID = B2, TOTALNOOFSEATS = 97, TYPENAME = AIRPLANETYPE_2).
AIRPLANE_2:(InstanceOf = AIRPLANE, AIRPLANEID = B3, TOTALNOOFSEATS = 300, TYPENAME = AIRPLANETYPE_2).

_ID:(InstanceOf = _AIRPLANE) :- _ID1:(_AIRPLANE = OHR).
```

For our preliminary results, we used the AIRPORTS database created into a class homework. This way we needed to transform that database into our input format. We show how we did this below.

From a Oracle Database to GDB

We designed a program which access Oracle database and transforms the information in it into DDL for GDB. This program first needs to restore the schema of each table in the database. This will allow it to get the skeleton of the future GDB. Then it needs to get all the instances of the Oracle database and fill the skeleton of GDB with these instances.

In order to get schemas of the tables we need to access several system-defined tables.

1. CAT - stores information about the names of all the tables in database.
2. COLS – stores information about the names of the columns in all the tables of the database.
3. USER_CONS_COLUMNS – for each foreign or primary key in the table it assigns the name of this constraint.
4. USER_CONSTRAINTS – says which exactly constraints are primary key and which are foreign keys and which constraints they reference.

We used the database from hw1 to test this program (links/createtbl.sql). Some of the entries in system tables for this database are given below:

CAT:

table_name	*table_type*
AIRPLANE	TABLE
AIRPLANETYPE	TABLE
AIRPORT	TABLE
CANLAND	TABLE
FARE	TABLE
FLIGHT	TABLE
FLIGHTLEG	TABLE
LEGINSTANCE	TABLE
SEAT	TABLE

COLS:

table_name	*column_name*
AIRPLANE	AIRPLANEID
AIRPLANE	TOTALNOOFSEATS
AIRPLANE	TYPENAME
AIRPLANETYPE	TYPENAME
AIRPLANETYPE	MAXSEATS
AIRPLANETYPE	COMPANY
AIRPORT	AIRPORTCODE
AIRPORT	NAME
AIRPORT	CITY

USER_CONS_COLUMNS:

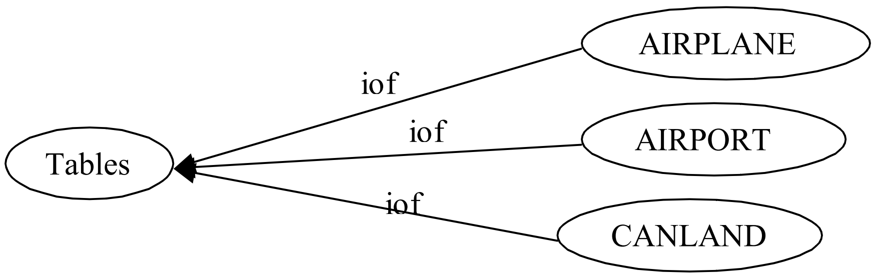
table_name	*constraint_name*	*column_name*	*position*
AIRPLANETYPE	SYS_C0056071	TYPENAME	1
AIRPORT	SYS_C0056072	AIRPORTCODE	1
CANLAND	SYS_C0056073	AIRPORTCODE	1
CANLAND	SYS_C0056073	TYPENAME1	2
CANLAND	SYS_C0056074	AIRPORTCODE	1
CANLAND	SYS_C0056075	TYPENAME1	1
AIRPLANE	SYS_C0056076	AIRPLANEID	1

USER_COSTRAINTS:

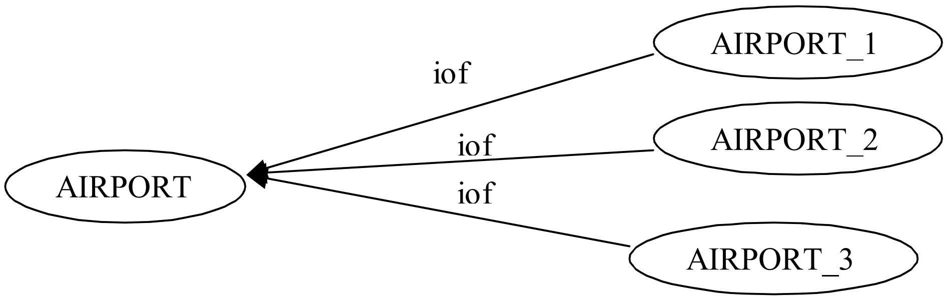
table_name	*constraint_name*	*constraint_type*	*r_constraint_name*
AIRPLANE	SYS_C0056076	P	null
AIRPLANE	SYS_C0056077	R	SYS_C0056071
AIRPLANETYPE	SYS_C0056071	P	null
AIRPORT	SYS_C0056072	P	null
CANLAND	SYS_C0056073	P	null
CANLAND	SYS_C0056074	R	SYS_C0056072
CANLAND	SYS_C0056075	R	SYS_C0056071

These tables are enough to restore the schemas of the tables. After this we need to get all the instances of the tables. We do it with simple query “SELECT * FROM *table_name*” for each table in the database.

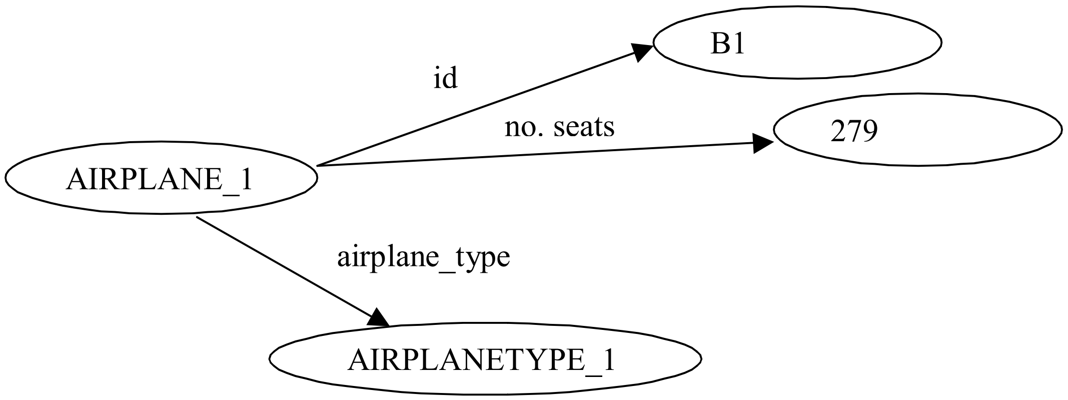
Now we explain how we transform schemas into skeleton of the future GDB and how we fill it with instances. First, we will generate a node for all tables called “Tables”. Then for each table we will create a parent node of “Tables” with the name of that table. The edge connecting them will be called “IOF” standing for “instance of”.



Then for each instance of the table we will create a new node, a parent of the node of the corresponding table:



Then for each such instance node we create edges with names "*attribute_name*" going to the node with particular value of that attribute in the case when this attribute is not a foreign key. If it is, then we connect this edge to corresponding instance node.



After all this information is gathered we transform it into DDL for Graph database.
Here is the code for the program: (Connect.java, Table.java, Constraint.java).
Here is the output file for the FLIGHTS database.

After all this information is gathered we transform it into DDL for Graph database.
Here is the code for the program: (Connect.java, Table.java, Constraint.java).
Here is the output file for the FLIGHTS database.

5. Conclusion and Future Work

Our interest in designing a new kind of database grew as a consequence of the problems arising in autonomous dynamic environments that are very common in a lot of scientific domains, and also business domains, Internet etc. Previous existing database systems don't take too much into account these problems, and they are not able to deal very well with changes. Also they do not offer a friendly machine learning environment that will allow the discovery of patterns in data, new concepts, explanations for some behavior etc. Thus, our goal was to design such a system that would be very useful for machine learning applications, and in the same time be able to efficiently and uniformly store any general information, that could also change in time.

Having this goal in mind, we design Graph Databases whose main building blocks are graphs representing uniformly data, knowledge, models and queries. The preference for graph structures came from the recent tendency of using graphical representations for data with the purpose of taking into account natural structure that data usually presents, as opposed to working with flat representations.

The system that we obtained is closed in spirit to OODB, but not the same. As opposed to GDB, we do not assume any encapsulation, but we pay a price by having more Ids, which is not too bad for our purpose of easily manipulating data in a natural declarative way.
GDB is also closed to Datalog, the main difference being in our option of using IDs (links) instead of foreign keys. This is also main difference from ORDBMs and somewhat from XML. However we do not lose anything by not using foreign keys, and we show before how we can simulate them using IDs.

Obviously, there are advantages and disadvantages in using GDB as a representaion/storage/reasoning/learning system, but for our purposes the disadvantages (more memory storage) are not essential and we are willing to pay this price toward the flexibility and unified view that we get. This is very important for the domains where we need to apply learning methods to extract information from data.

We are aware that a lot of the existent data resides in relational databases, and in order to be able to reuse this data, an important part in our project is focused on the translation from relational databases format to our GDB input format.

Regarding future work, our short-term objective is to add more features to the current implementation that is still in the initial phase. We couldn't implement for this project all the things that we showed that can be easily incorporated in our system. We plan to include special procedures for handling updates, changes, adding new concepts to data graph etc. Our current implementation was mainly focused on the general design, and also on the declarativity issue that we were able to achieve as opposed to other system based on graph.

As long term goal, we will use the DBG structures for doing learning on them, in the hope of finding interesting information from structural data and knowledge.

6. References

[1.] D. J. Cook and L. B. Holder (2000) *Graph-Based Data Mining*, IEEE Intelligent Systems, 15(2).

[2.] L. Getoor, Friedman, N., D. Koller, B. Taskar (2001). *Probabilistic Models of Relational Structure.* International Conference on Machine Learning, Williamstown, MA.

[3.] F. V. Jensen (2001). *"Bayesian Networks and Decision Diagrams".* Springer.

[4.] M. I. Jordan (ed). (1998) .*"Learning in Graphical Models".* MIT Press.

[5.] J. Pearl. (2000) *"Causality".* Cambridge.

[6.] J. Reinoso, A. Silvescu, V. Honavar and S. Gadia (2002).*Ontology-Assisted Information Extraction and Integration.* In preparation.

[7.] A. Silberschatz, H. Korth and S. Sudarshan (1996). *Database System Concepts*. McGraw-Hill Computer Science Series.

[8.] J. Sowa (2000). *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. New York: PWS Publishing Co.

[9.] M. Young (2001). *XML Step by Step*. Second Edition.