GRACE: A Graph Database System

Srinath Srinivasa Mistry Harjinder Singh

International Institute of Information Technology
26/C, Electronic City
Bangalore 560100
India
{sri, mistry.harjinder}@iiitb.ac.in

Abstract

GRACE is a database management system for managing data that is in the form of labeled undirected graphs. A large amount of data generated today, like road maps, protein-structures, citation-data, etc. are in the form of undirected labeled graphs. Also, problems involving such data require new forms of queries that are run on the structural properties of the graph. Finding structural patterns in a protein database is a good example. Relational DBMS are not suitable for managing structural queries. GRACE introduces a new data model for graph data and a query language called Safari, which supports both attribute and structural searches.

1 Introduction

Graph databases are gaining importance in many applications like molecular biology. The main requirement is to manage the data in the form of graphs.

The applications should be able to add graphs to a database and retrieve graphs based upon a given criterion. For example, a query may ask to retrieve graphs, having similar structure as the given query-graph. Further, the graphs can also be modified. For example, nodes and edges can be added or deleted from the existing graph.

GRACE is a Graph Database, which enables its users to manage the graph-data. The users can add graphs to a database. The graphs can be retrieved based upon the properties of the graphs. The properties of a graph include the attributes associated with a graph and the structure of the graph.

The current version of GRACE and its features were first proposed in [1].

International Conference on Management of Data
COMAD 2005b, Hyderabad, India, December 20–22, 2005

2 GRACE Model

A GRACE database G is a collection of member graphs, $G = \{G_1, G_2, G_3, \ldots, G_n\}$. Each member graph is a labeled graph of the form $G_i = \{V, E, V_L, \lambda, V_N, \delta\}$. Here, V is a set of vertices. E is a set of edges. V_L is a set of vertex labels. V_N is a set of vertex names. $\lambda : V \rightarrow V_L$ is a vertex labeling function and $\delta : V \rightarrow V_N$ is vertex naming function.

The nodes and graphs can be assigned labels. A label defines a type. All the nodes having the same label belong to the same type. Note that for every node in a graph, the combination of label and name must be unique. For example, in a graph of cities, a node may be specified as follows: City_Bangalore, City_Mumbai, City_Pune, etc.

Each node in the above example, has a label 'City', i.e. each node belongs to same type called 'City'. However, each node has unique name, which differentiates between two instances of the same type. Here, Bangalore is an instance of type City.

In GRACE, an attribute can be associated with a graph, a node or an edge. An attribute consists of an attribute-name and its corresponding value. In current implementation of GRACE, the type of the attribute is not considered. It supports string attributes only.

Consider the following graph (specified in the modified DOT format, see: Appendix):

```
graph cities_metro [country=India]
{
  city_Bangalore [area=1000sqkm, population=6m];
  city_Mumbai [area=2000sqkm, population=12m];
  city_NewDelhi [area=2500sqkm, population=11m];
  ;
  city_Bangalore--city_Mumbai [distance=1100km,
   journey=23hrs];
  city_Mumbai--city_NewDelhi [distance=990km,
   journey=18hrs];
  city_NewDelhi--city_Bangalore [distance=2700km,
   journey=36hrs];
}
```

Here, the graph cities_metro has the attribute with name=country and value=India.

Similarly, the node city_Mumbai has two attributes:

1. attribute-name = area; value = 2000sqkm
2. attribute-name = population; value = 12m

The edge city_Mumbai–city_NewDelhi has two attributes:

1. attribute-name = distance; value = 990km
2. attribute-name = journey; value = 18hrs

2.1 Schema Graph

A collection of graphs can be described by a Schema. However, in GRACE, schema is also a graph. So a member graph can become a schema graph and refer to a group of graphs in the database. A member graph is called a Schema Graph, if and only if, there is at least one node in that graph, which refers to another member graph in the database.

In schema graph, other member graphs are referenced through nodes. The nodes in the schema graph can refer another member graph in the following ways:

Using graph ID: Each graph in the database is given a unique ID called Graph ID(GID). This gid can be used as a reference. Consider a case when a member graph with gid = x, is referenced by a node N in a schema graph. Then a 'logical' link between the node N and member graph (gid=x) can be established by having gid=x as an attribute of node N.

Using label-name combination: The combination of a graph's name and label has to be unique across the database. By having this pair of label and name as attributes of the referencing node, a 'logical' link between the referencing node and the referenced member graph can be established.

Using a Safari query: A Safari query returns graph structure. Nodes in the schema graph can use such queries to reference other graphs.

Note that the first two methods are static methods and the last method is dynamic method of referencing other graphs from within a schema graph.

Insertion of a schema graph may fail, if any of its nodes try to reference a graph which does not exist. This maintains referential integrity. The graph referenced by a node in a schema graph may in turn be a schema graph itself. This can continue to any level and self references are also allowed. A schema can refer to itself as one of its nodes.

2.2 Default Graph

By default, all member graphs are referenced by a schema graph, called Default Graph. It is the very first graph created when a database is created and it cannot be removed from the database. It is a disconnected graph, i.e. no edge exists between any two nodes. However, edges can be inserted.

Initially, when a database is created, it contains only one graph, namely Default graph. This graph has one node referencing to itself. When a graph is added in a database, a node is added in the default graph, which will refer to the newly added graph.

3 SAFARI: Grace Query Language

A graph is specified in modified form of DOT format developed under the Graphviz project at $AT\&T$.[1] Safari is a query language that can manipulate graph objects specified in the modified DOT form and manage their storage and retrieval in databases. There are four types of Safari Constructs:

1. Graph Modification Constructs: These constructs enable the users to add or remove nodes and edges to a graph. The attributes of the nodes, edges and graph can be updated.
2. Graph Retrieval Constructs: These constructs return a subgraph of an input graph based on a given condition.
3. Database Modification Constructs: These constructs add and remove graphs to and from databases.
4. Database Retrieval Constructs: These constructs search the database for graphs matching the specified condition and return a member graph or a dynamically created meta-graph. A meta-graph is a pure schema graph, where each node represents a member graph.

Currently, GRACE does not support the graph modification constructs of Safari.

3.1 Graph Retrieval Constructs

The main graph retrieval construct is 'selecton' operator (pronounced as select on). selecton will give a subgraph of the input graph, as its output. It has the following syntax:

$$\sigma_o\langle condition\rangle[\langle graphref\rangle]$$

The selecton takes two inputs: one is the graph, on which it will operate and the other is the condition for matching. In the syntax, the condition is specified first. The input graph is specified by the reference

[1] www.http://www.research.att.com/crg/graphviz/info/lang.html

and it is optional. If no input graph is specified, the Default graph of the current database is taken as an input graph.

The input graph reference can be specified by one of the following ways:

1. Using a "session variable."
2. Providing an inline DOT formatted graph.
3. Placing another Safari query output.

Note that, in the absence of input-graph, the Default graph of the current database will be taken as input. For example,

```
selecton nodeattr('state','kerala')
IndianCities;
```

This query will give all the cities of state Kerala, from the graph of Indian cities. Note that the input graph is specified by a session variable 'IndianCities'. Session variables are temporary variables holding graph structures as part of a client session.

3.2 Database Modification Constructs

The 'insert' construct provides the facility to add a graph to a database. It has the following syntax:

$$insert\langle graphref\rangle$$

where graphref is a reference to a graph. The input graph will be added to the current database.

The 'use' construct selects a database. It has the following syntax:

$$use\langle dbname\rangle$$

It selects the specified database as the current database. The index structures and default graph of the selected database will be loaded into memory.

The 'create' construct is used to create a new database. It has the following syntax.

$$create\langle dbname\rangle$$

where dbname is the name of the database being created.

3.3 Database Retrieval Constructs

The selectin and selectgraph are two main constructs that search the database for the graphs that match the input condition.

selectin has the following syntax:

$$\sigma_i\langle condition\rangle[\langle graphref\rangle]$$

First parameter is a logical condition either on the attributes or on the structure of the graph. The graphref is a reference to the graph. The selectin operator works as follows:

1. The graph specified in the last parameter is used as a schema graph and all the graphs referred by it are searched for the specified condition.
2. The schema graph is then pruned to remove all nodes that either do not refer to other graphs, or refer to the graphs that do not match the condition. When a node is removed, its incident edges are also removed.
3. The pruned schema graph is then returned.

For example,

```
selectin nodeattr('author','codd')
dbpapers;
```

This query will return a schema graph, in which each node will refer to a graph, having atleast one node with attribute: 'author' = 'codd'. Here, 'dbpapers' is a session variable.

The selectgraph operator selects a member graph from the database. It has the following syntax:

$$\sigma_g\langle condition\rangle[\langle graphref\rangle]$$

The selectgraph matches the condition against each graph referred by the schema graph (graphref). If only one graph satisfies the condition, then it will be returned. When two or more graphs satisfy the condition, then a schema graph will be returned, in which each node will refer to a graph. For example,

```
selectgraph nodeattr('__name','mumbai');
```

This query will return the graph whose name is 'mumbai'. Here, input schema graph is not specified, so the default graph will be taken as the input and the entire database will become the search space.

3.4 Condition Specification

The properties of a graph include the following:

1. The attributes of nodes, edges and the graph itself.
2. The structure of the graph.

Both of these properties can be used in the 'condition' parameter of the Safari queries.

The use of the 'attributes' in the condition has been shown in the previous examples.

1. nodeattr(attrname, attrvalue) returns all the nodes which have an attribute matching first parameter and whose value matches second parameter. If two or more nodes are returned, any edges among them are also returned.
2. edgeattr('attrname','attrvalue') returns all edges which have an attribute matching the first parameter, whose values matches second parameter. When an edge is returned, the corresponding nodes are also returned.

The use of structure property in the 'condition' parameter can be done using the following functions:

1. similar(graphref): returns all the graphs, which are 'structurally similar' to the specified 'graphref'. Here, the search space is the set of graphs referred by the specified schema graph.

   Two graphs are structurally similar, if the structural distance between them is smaller than a threshold value. The structural distance is the edit distance, which is defined as the number of node/edge edit operations that are required to transform one graph to another. The threshold value is a configurable parameter in GRACE. For example,

   ```
   selectin similar(fructose);
   ```

   This query will return all the graphs in the database, which are similar to the graph referred by the session variable, 'fructose'. Note, here no schema graph is specified, so the default graph of the current database is taken as an input.

2. substructure(graphref): returns all the graphs which are likely to have the graph (referred by graphref) as a subgraph.

GRACE uses the Label Walk Index(LWI), to perform the structural queries: similar() and substructure(). The LWI was proposed in [1].

4 GRACE Clients

Current version of GRACE works with the following clients:

1. Telnet Client: The user can open a telnet session and connect to the GRACE server. The commands given at the GRACE prompt will be processed at the server and the response will be given back to the user.

2. PHP interface: GRACE has PHP interface, which enables the user to connect to the GRACE through PHP code. This makes it possible to use GRACE in three-tier application architectures.

References

[1] S. Srinivasa, M. Maier, M. R. Mutalikdesai, K. A. Gowrishankar, P. S. Gopinath. LWI and Safari: A New Index Structure and Query Model for Graph Databases. *In Proc. 11th International Conference on Management of Data (COMAD 2005), Goa, India, Jan 2005.*

5 Appendix: Modified Dot Format

The following is the modified dot format:

```
graph [graphlabel_]graphname [graphattribute]
{ [nodesection]
 ;
[edgesection]
}
```

Every graph specification starts with a 'graph' keyword. The graphlabel is optional, but the graphname is compulsory. The graphattribute is optional. It has the following form:

$$[attributename = attributevalue]$$

The attribute is specified within square brackets. Here attributename is the name of the attribute and attributevalue is the value of the attribute.

Within braces, there is an optional node section, which specifies a set of nodes and an optional edge section, which specifies a set of edges.

5.1 Node section

Each node specification has the following form:

$$[nodelabel\_]nodename[nodeattribute];$$

Here, nodelabel is optional. The nodeattribute is optional. It has the same form as that of the graphattribute, discussed above. The node specification ends with a semicolon(';'). To seperate two node-specifications, a semicolon(';') is used.

At the end of the specification of all the nodes, a semicolon(';') is added to mark the end of the node section.

5.2 Edge section

Each edge specification has the following form:

$$[nodelabel\_]nodename--[nodelabel\_]name[edgeattribute];$$

The edge is specified by two nodes, seperated by '–'(two minus) operator. The edgeattribute is optional. It has the same form as that of the graphattribute, discussed above. The edge specification ends with a semicolon(';'). To seperate two edges, a semicolon(';') is used.

An example graph decription in modified dot format:

```
graph aliphatic_methane [category=hydrocarbon]
{
H_0 [name=h2,nodename=Hydrogen];
C_0 [nodename=carbon] ;
;
H_0--C_0 [bond=single] ;
}
```