|  |
| --- |
|  |
| GRaph DataBase |
| grdb User’s Guide |
|  |
|  |
| **5/29/2017** |

|  |
| --- |
|  |

# Interactive Use

When the database starts up you get a command prompt:

**fwmiller@ubuntu:~/graph-database/src/cli$ ./db**

**Graph Database**

**(C) Frank W. Miller**

**grdb>**

Most commands and options have two forms, their long form and an abbreviation. For example, the graph command can be issued using either **graph** or **g**. See the Appendix for the CLI syntax summary.

## Creating a Graph

When you see the **grdb>** prompt you can enter a command to the database. When the database starts up, it is empty, i.e. it contains no graphs. To create a new graph and then print the list of graphs, type the following commands:

**grdb> g n**

**grdb> g**

**>0.0:({1},{})**

The first command creates a new graph. **g** is the graph command and **n** represents the create new graph operation. If you just type **g** as shown in the second command, the list of graphs currently in the database is printed.

There is only one line listed since we just created our first graph. The graph number has a number that is composed of two parts separated by a decimal point. The first part is the graph number and the second part is the component number. For a new graph numbered 0 with a single component numbered 0, the value 0.0 is displayed.

The cursor, **>**, at the beginning of the line indicates that it is the current graph and component. After the colon, the component is presented in graph notation, i.e. a pair consisting of a set of vertices and a set of edges: ( *V*, *E* ). In this example a new graph is created with a single vertex that has an id of 1 and an empty set of edges. Note that the **g** command does not dump the tuples associated with the graph vertices and edges.

## Adding Edges

To build the graph up, you add edges to components. For example, the following command adds the edge, (1,2), to our current graph 0.0:

**grdb> g e 1 2**

**grdb> g**

**>0.0:({1,2},{(1,2)})**

The component 0.0 now contains an additional vertex, 2, and the edge, (1. 2). The **g e** command requires that you specify two vertex ids as the endpoints for the edge, in this case, 1 and 2. *There is an additional restriction that one of the vertex ids must already exist in the current component.* If one of the vertex ids is not in the component, it will be created and added to the vertex set for the current component. Some additional examples:

**grdb> g e 1 3**

**grdb> g e 3 2**

**grdb> g e 5 2**

**grdb> g**

**>0.0:({1,2,3,5},{(1,2),(1,3),(3,2),(5,2)})**

**grdb> g e 8 9**

**At least one vertex must exist in component**

**grdb>**

## Adding Schemas

Schemas define the types of data associated with the vertices and edges of a component. Each component has two schemas 1) a schema for all its vertices and 2) a schema for all its edges. You can create these schemas as follows:

**grdb> g s**

**grdb> g s v int i**

**grdb> g s e int j**

**grdb> g s**

**>component 0**

**Sv = [INT:i]**

**Se = [INT:j]**

**grdb> g s v float k**

**grdb> g s**

**>component 0.0**

**Sv = [INT:i,FLOAT:k]**

**Se = [INT:j]**

**grdb>**

The **g s** command prints a summary of the schemas for all components of all the graphs. Schemas are added by using a command like **g s v int i**, which will cause an attribute to be added to the vertex schema of the current component 0.0 that is called **i** and has type **int**. The command **g s e int j** is similar adding an attribute to the edge schema of the current component 0 that is called **j** and has type **int**.

The command **g s v float k** adds a second attribute to the vertex schema. When adding a sequence of attributes they are each concatenated to the end of the schema attribute list. While theoretically, there should be no ordering of the attributes, the implementation imposes an ordering by its design.

There is no limit on the number of attributes imposed by the implementation other than the size of main memory.

The base data types that can be used in schemas are summarized as:

|  |  |  |
| --- | --- | --- |
| **Type** | **CLI syntax** | **Size (bytes)** |
| Character | **char** | 1 |
| String | **varchar** | 256 |
| Boolean | **bool** | 1 |
| Signed Integer | **int** | Machine dependent |
| Floating point | **float** | Machine dependent |
| Double precision floating point | **double** | Machine dependent |
| Date | **date** | 10 |
| Time | **time** | 8 |

## Enums

grdb includes an enumerated type that can be used as attributes for vertices and edges. An enumerated type is defined as follows:

**grdb> enum relatives father mother brother sister son daughter**

**grdb> enum**

**relatives (father,mother,brother,sister,son,daughter)**

**grdb>**

Once defined, the enum **relatives,** can be used anywhere a base type can.

## Tuples

User data is contained in tuples that are attached to the vertices and edges of each component. The user data is defined by a component’s schemas. The database will maintain consistency in the schemas and tuples associated with each component. That is, if you change the schema, all the tuples will be updated appropriately. If you add an edge, its tuples takes on default values according to the current defined schemas.

**grdb> g n**

**grdb> g s v int i**

**grdb> g s v relatives r**

**grdb> g s e double k**

**grdb> g s**

**>component 0.0**

**Sv = [INT:i,relatives:r]**

**Se = [DOUBLE:k]**

**grdb> g e 1 2**

**grdb> g e 2 3**

**grdb> g e 3 1**

**grdb> g**

**>0.0:({1,2,3},{(1,2),(2,3),(3,1)})**

**grdb> g t**

**({1[0,father],2[0,father],3[0,father]},{(1,2)[0.00],(2,3)[0.00],(3,1)[0.00]})**

In this example, a pair of attributes is set for the vertex schema and a single attribute is set for the edge schema. A series of edges is then added to create a simple three vertex cycle. The last command **g t** displays the current graph but also presents the user data in the form of the tuples associated with the vertices and edges. Each of the vertices has two values associated with it denoted by the list of values in square brackets following the vertex id. The first is and **INT** and the second is the enum **relatives** that we just defined according to the vertex schema. Likewise, each edge has a single **DOUBLE** value associated with it.

To modify a tuple value, you use the **g t** command. Consider the following example:

**grdb> g n**

**grdb> g s v int i**

**grdb> g t**

**>0.0:({1[0]},{})**

**grdb> g t 1 i 5**

**grdb> g t**

**>0.0:({1[5]},{})**

Here a new graph is created a single attribute is added to the schema vertex. Then the command **g t 1 i 5** is used to set the value of the integer I value in the vertex 1 to the value 5. You can then see by dumping the tuples that the value of tuple for the vertex 1 has been updated. An edge tuple can be updated similarly:

**grdb> g n**

**grdb> g e 1 2**

**grdb> g s e varchar s**

**grdb> g t**

**>0.0:({1,2},{(1,2)[""]})**

**grdb> g t 1 2 s “this is a test”**

**grdb> g t**

**>0.0:({1,2},{(1,2)["this is a test"]})**

For varchar data values, the character string is enclosed in double quotes.

# Scripting

The database can be driven using scripts. The scripts are sequences of the commands that can be issued to the CLI. Consider the following script **alltypes** that creates a graph with tuples that have examples of all the base types.

**g n**

**g e 1 2**

**g s e char i**

**g s v bool k**

**g s e int l**

**g s v float m**

**g s e double n**

**g s v date d**

**g s e time t**

**g s v varchar s**

**g s**

**g t**

**quit**

This script can be issued to the database by executing the database as follows:

**$ ./db < alltypes**

that gives the following output:

**g n**

**g e 1 2**

**g s e char i**

**g s v bool k**

**g s e int l**

**g s v float m**

**g s e double n**

**g s v date d**

**g s e time t**

**g s v varchar s**

**g s**

**>graph 0**

**Sv = [BOOL:k,FLOAT:m,DATE:d,VARCHAR:s]**

**Se = [CHAR:i,INT:l,DOUBLE:n,TIME:t]**

**g t**

**({1[FALSE,0.00,08-27-2016,""],2[FALSE,0.00,08-27-2016,""]},{(1,2)['',0,0.00,00:00:00]})**

**quit**

This facility can be used to program graph problems by creating and then executing operations on programmed graphs.

# Command Summary

**help|h|?**

**quit|q**

**about|a**

**graph|g**

**|**

**+--- new|n**

**|** Create a new graph and give it the next graph id in sequence starting with zero.

**|** The new graph contains only a single vertex with no edges and no schemas.

**|**

**+--- enum** *<id> <id list>*

**|** Create a new enumerated type called *<id>* and associate all the values in the *<id list>*

**|** with it. The *<id list>* is a space separated list of identifiers that represent the

**|** individual enumerated values.

**|**

**+--- edge|e** *<vertexid1> <vertexid2>*

**|** Add the edge (*vertexid1*, *vertexid2*) to the current graph. At least one of the vertexid’s

**|** must already exist in the graph. If a schema is set for the edges, the tuple for the new

**|** edge takes on default values.

**|**

**+--- schema|s**

**| |**

**| +--- vertex|v** *<type> <id>*

**| |** Add an attribute to the vertex schema for the current graph. The new attribute will

**| |** have the specified *<type>* and be assigned the name *<id>*. All vertices in the current

**| |** graph will have their tuples updated with the new attribute set to a default value.

**| |**

**| +--- edge|e** *<type> <id>*

**|** Add an attribute to the edges schema for the current graph. The new attribute will

**|** have the specified *<type>* and be assigned the name *<id>*. All edges in the current

**|** graph will have their tuples updated with the new attribute set to a default value.

**|**

**+--- tuple|t**

**| |**

**| +---** *<vertexid>**<id> <value>*

**| |** Set the tuple attribute named *<id>* for the vertex *<vertexid>* to the specified *<value>*

**| |**

**| +---** *<vertexid1>**<vertexid2> <id> <value>*

**|** Set the tuple attribute named *<id>* for the edge (*<vertexid1>*, *<vertexid2>*) to the

**|** specified *<value>*

**|**

**+---** *<graphid>*

Set the current graph to *<graphid>*