Introduction

This document is a guide about how to install and use the Rogas system.

Rogas not only can provides a high-level declarative query language to formulate analysis queries, but also can unify different graph algorithms within the relational environment for query processing.

This document includes the following parts:

How to Install Rogas

The Rogas system has the following dependencies:
- Python 2.7
- Tornado: http://www.tornadoweb.org/en/stable/
- Postgresql: https://www.postgresql.org/
- Psycopg: http://initd.org/psycopg/
- Graph-tool: http://graph-tool.skewed.de/
- SNAP: http://snap.stanford.edu/snappy/index.html
- NetworkX: http://networkx.github.io/

The following instruction describes every step to install the dependencies and provides solutions of some common errors that you may encounter.

1. Check the Python version

```
$ python2.7 --version
Python 2.7.9
```

If your python version is not 2.7, then check if you system has python 2.7
(Normally, Ubuntu install python2.7 by default)

```
$ ls /usr/bin/python*
/usr/bin/python   /usr/bin/python2.7  /usr/bin/python3.4
/usr/bin/python2  /usr/bin/python3
```

If your system does'n have python, please download from the Python official website
(https://www.python.org/)

2. Install Tornado (http://www.tornadoweb.org/en/stable/)

```
$ pip install tornado
```

3. Install PostgreSQL (https://www.postgresql.org/download/linux/ubuntu/)

```
$ sudo apt-get install postgresql-9.4
```

If you encounter such an error: "psql: FATAL: role "minjian" does not exist"

```
$ sudo su postgres
$ psql
postgres=# create user minjian(your username) superuser;
```

If you encounter such an error: "psql: FATAL: database "minjian" does not exist"

```
$ psql postgres
postgres=# create database minjian;
```

4. Install Psycopg (http://initd.org/psycopg/)

$ sudo apt-get install python-psycopg2

5. Install Graph-tool (https://graph-tool.skewed.de/download#packages)

$ sudo gedit /etc/apt/sources.list

At the end of the sources.list append:

deb http://downloads.skewed.de/apt/DISTRIBUTION DISTRIBUTION universe

deb-src http://downloads.skewed.de/apt/DISTRIBUTION DISTRIBUTION universe

(Replace the "DISTRIBUTION" with the codename of your system, for example 15.04 is called vivid, you can find the codename by run the command "lsb_release -a")

$ sudo apt-get install python-graph-tool

If you encounter such an error "python-graph-tool : Depends: libstdc++6 (>= 5) but 4.9.2-10ubuntu13 is to be installed", then you need to update your gcc and g++ compiler.

$ sudo add-apt-repository ppa:ubuntu-toolchain-r/test && apt-get update

$ sudo apt-get install gcc-5 g++5

6. Install SNAP (http://snap.stanford.edu/snappy/index.html)

go to the http://snap.stanford.edu/snappy/release/

download the "snap-3.0.2-3.0-centos6.5-x64-py2.6.tar.gz"

$ tar zxvf snap-3.0.2-3.0-centos6.5-x64-py2.6.tar.gz

$ cd snap-3.0.2-3.0-centos6.5-x64-py2.6/

$ sudo python setup.py install

(Be sure the "python" bin you used is Python2.7)

7. Install NetworkX (https://networkx.readthedocs.io/en/stable/index.html)

$ pip install networkx

8. Check the three graph tools are installed successfully

$ python

>>> import snap

>>> import networkx

>>> import graph_tool.all

(No errors, then they are all installed correctly)

The following is about the solutions of some common errors, if you installed the graph tools successfully, you can skip the following steps and goto the “How to Run” section.

If you encounter such an error:

Boost.Python.ArgumentError: Python argument types in
graph_tool.libgraph_tool_core.openmp_set_schedule(str, int)
did not match C++ signature:
openmp_set_schedule(std::__cxx11::basic_string<char, std::char_traits<char>,
std::allocator<char> >, int)

Then the pre-compiled package of graph_tool doesn't match with your current c/c++ lib. You need to download the source code of the graph-tool and compile it by yourself. The source code can be found in this link ("https://graph-tool.skewed.de/download")

$ cd graph-tool-2.19

$ ./configure

$ make

$ sudo make install

Note: during the process of "./configure", you might need to install or upgrade other dependencies mentioned in the graph_tool website. The following is the steps about how to install or upgrade some of dependencies:

Upgrade the Boost Libraries:

goto "https://sourceforge.net/projects/boost/files/?source=navbar" and download the latest boost lib, perhaps you need to install the following dependencies before the boost installation.

```
$ sudo apt-get install mpi-default-dev
$ sudo apt-get install libicu-dev ou
$ sudo apt-get install python-dev
$ sudo apt-get install libbz2-dev
```

Install the Boost Libs

```
$ cd boost_1_62_0/
$ ./bootstrap.sh
$ sudo ./b2  install
```

Install CGAL, Sparsehash, Cairomm and Pycairo:
(Note:Different versions of Ubuntu may correspond different names of these packages, if you cannot locate these packages, please google it like "Ubuntu Cairomm" to find the corresponding package name)

```
sudo apt-get install libcgal-dev
sudo apt-get install libcgal-demo
sudo apt-get install libsparsehash-dev
sudo apt-get install libcairomm-1.0
sudo apt-get install python-cairo-dev
```

If encounter such an error: "fatal error: CGAL/Periodic_3_Delaunay_triangulation_traits_3.h: No such file or directory"

```
goto "https://github.com/CGAL/cgal/releases" and download the lastest version of CGAL
$ cd CGAL-4.9
$ make
$ sudo make install
```

It is quite difficult to install graph_tool, so please be careful to read the instruction of the link "https://graph-tool.skewed.de/download" about how to install graph_tool. Many unexpected issues often occur during the installation, if the issues haven't been mentioned above, please Google them to seek solutions.

How to Run Rogas

1. Clone the Rogas Git Repo

```
$ git clone https://github.com/CornucopiaRG/Rogas.git
```

2. Create the sample database

```
$ psql
# create database stack;
# \q
$ psql stack < stack.bak[1]
$ psql stack
```

There are four tables in the stack database:

```
                List of relations
 Schema |    Name     | Type  |  Owner
--------+-------------+-------+---------
 public | answer      | table | minjian
 public | labelled_by | table | minjian
 public | question    | table | minjian
 public | tag         | table | minjian
(4 rows)
```

3. Modify the config.py:

```
$ vim rogas/config.py
```

According to your DB information, modify the config.py like the figure on the right.

```
#information to connect to database
DB = "stack"
DB_USER = "minjian"
DB_PASSWORD = "111"
DB_PORT = 5432

#each page has 10 records
PAGE_MAX_NUM = 10

#whether graph_tool support openmp
IS_GRAPH_TOOL_OPENMP = True
```

4. Start the system:

```
$ python run.py
```

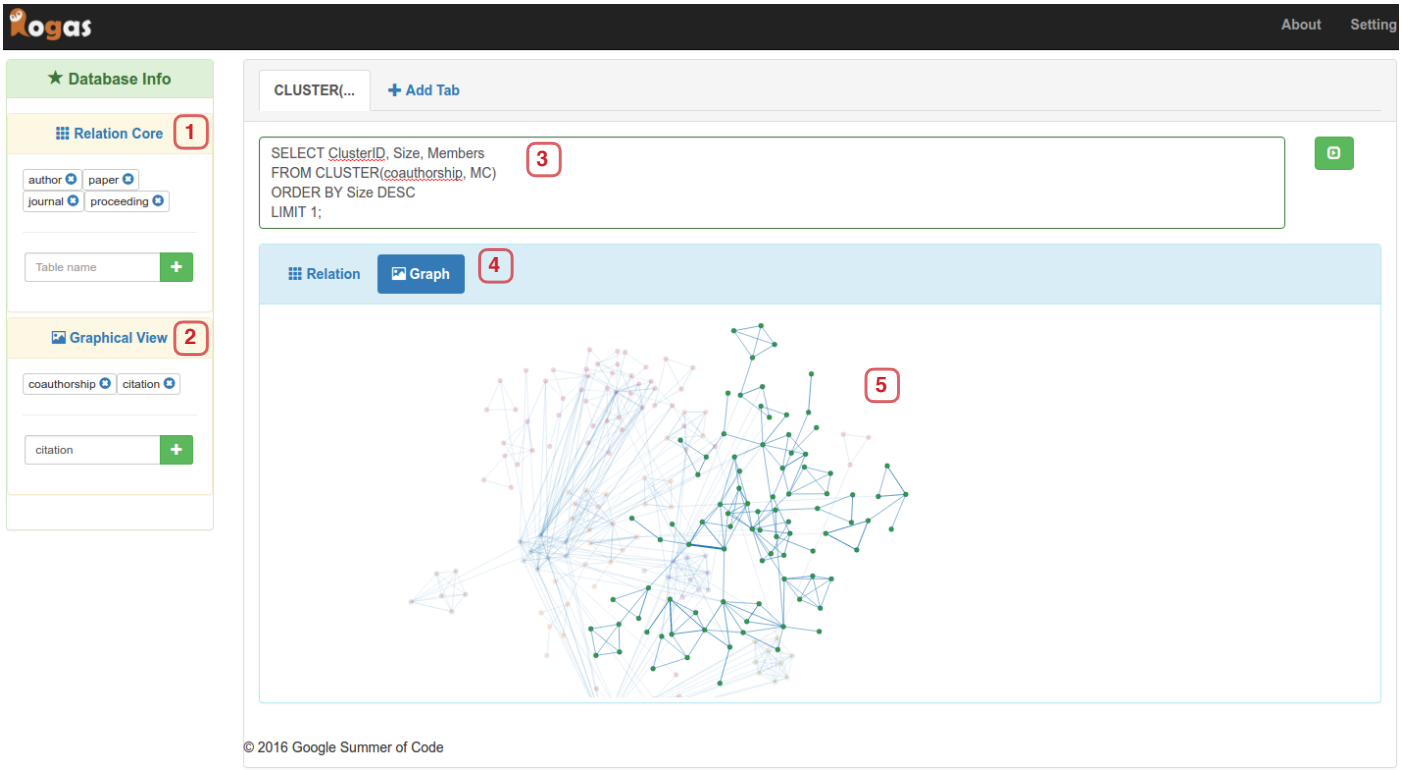
Open the browser by entering http://localhost:12345/

---

[1] stack.bak is a small part of the StackOverflow data from http://snap.stanford.edu/proj/snap-icwsm/

How to Use Rogas

1. GUI Overview



(1) Relational Core: Add a tag by typing the table name. Click the “Relation Core” label and the tags below to view more general information of tables.

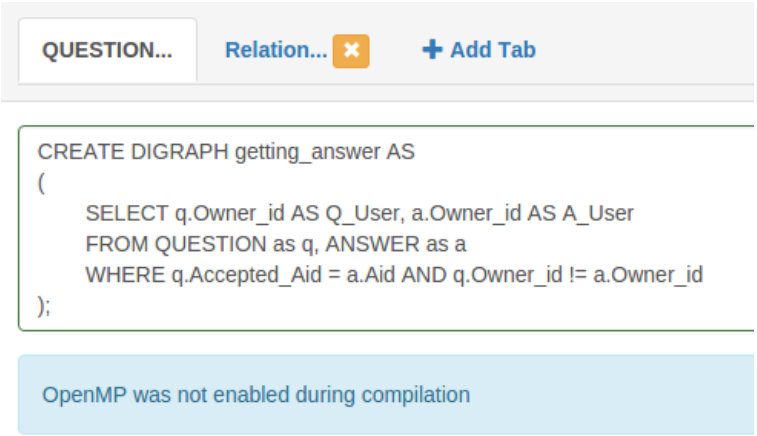
(2) Graphical View: Add a tag by typing the graph name. Click the “Graphical View” label and the tags below to view more general information of graphs.

(3) Query Input: Write a query here, the query can be a regular relational query or a graph query with the graph operators. Click the green “run” button on the right or press “Ctrl+Enter” to run the query.

(4) Result Tabs: If the query is a regular relational query, then only the “Relation” result tab will be shown. If the query is a graph query, then the “Relation” and “Graph” result tabs will be shown. You can click one of the result tabs to view the corresponding result.

(5) Result Panel: You can view the results here. Relation results will be shown as a table and graph results will be shown as vertices and edges.

Note: If you encounter such an error: “OpenMP was not enabled”, it means your compiler doesn’t support OpenMP, then please goto rogas/config.py and modify “IS_GRAPH_TOOL_OPENMP” as False.



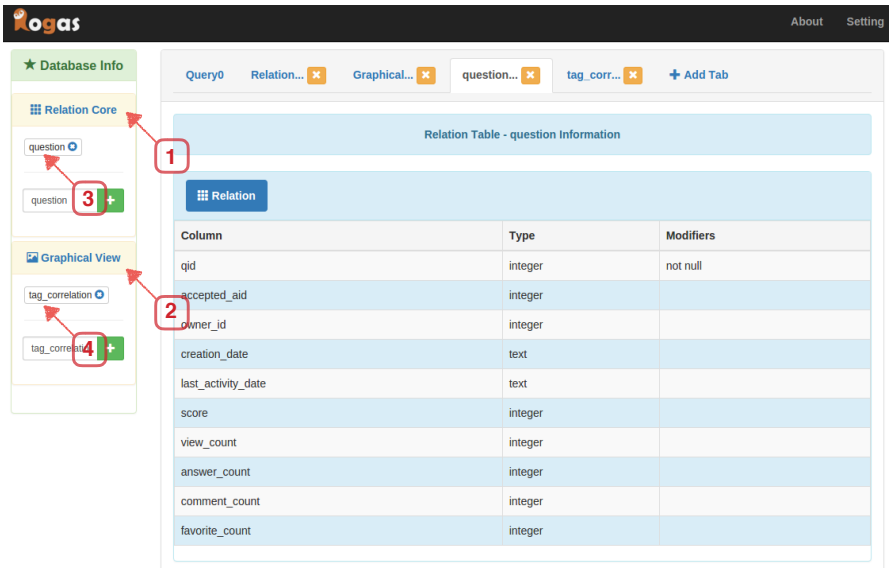
2. View General Info

(1) Click “Relation Core”, it will show all the table general information in the database.

(2) Click “Graphical View”, it will show all the graph general information in the database.

(3) Click the “question” tag, it will show the schema information of the question table.

(4) Click the “tag_correlation” tag, it will show the graph statistics of the tag_correlation graph.



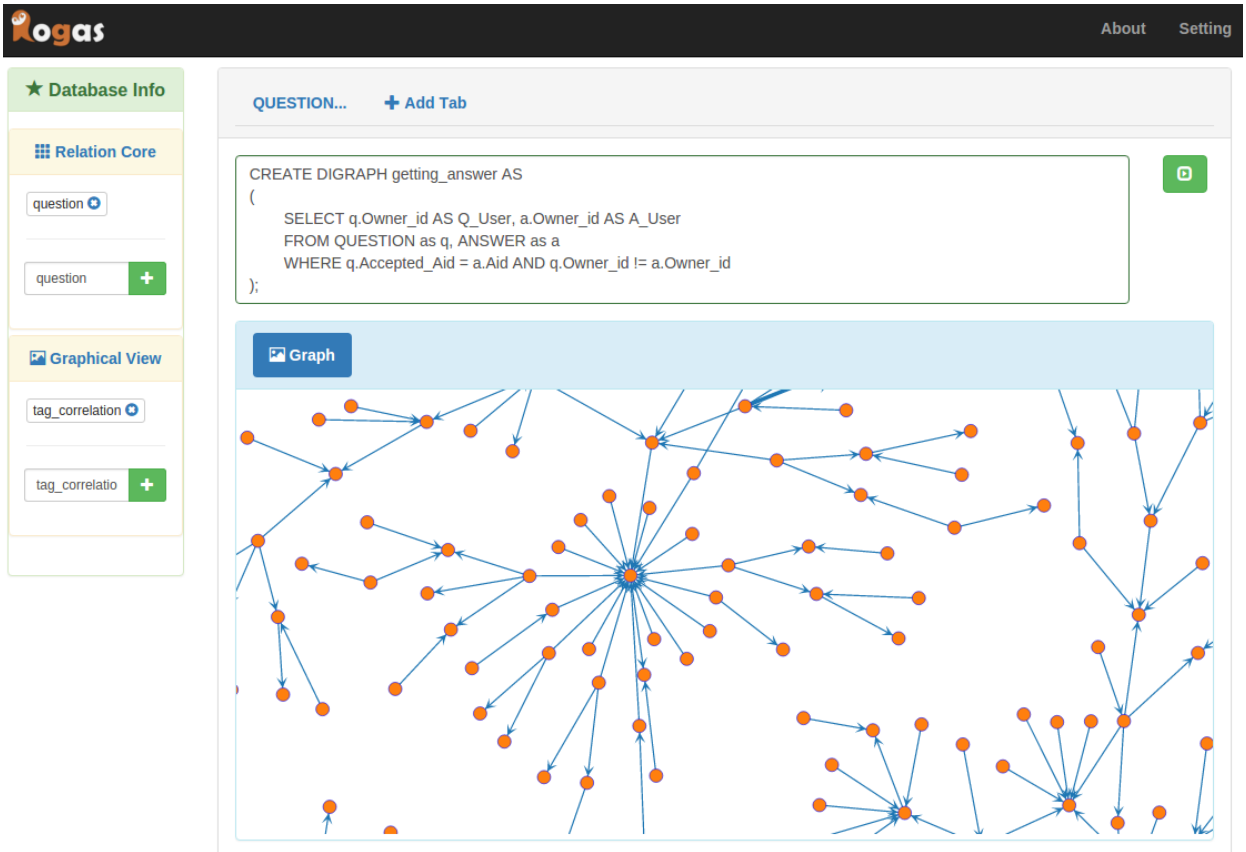
3. Create and Drop Graphs

Syntax of creating graphs:

CREATE <graph type > <graph name> AS (mapper);

<graph type> := UNGRAPH | DIGRAPH

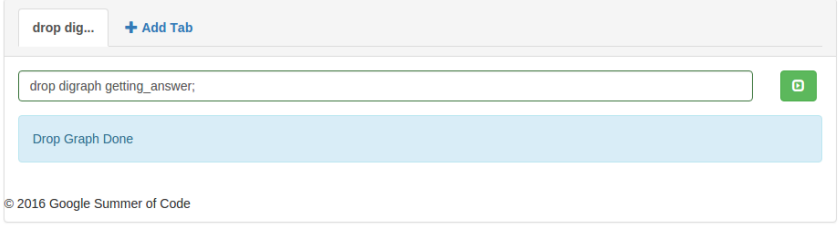
A mapper query has the form of SELECT-FROM-WHERE, which extracts an edge list from relations in the underlying databases for graph construction.



Syntax of dropping graphs:

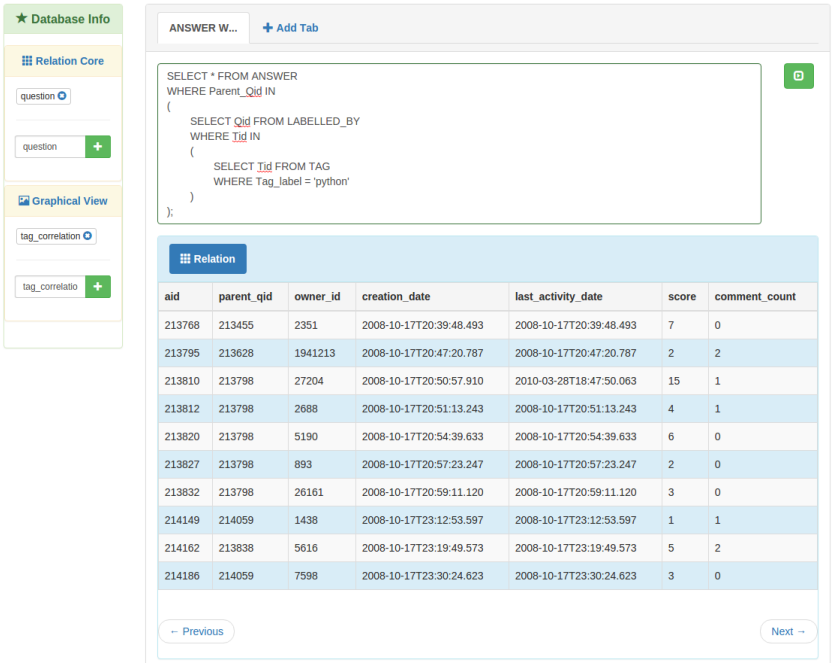
DROP <graph type > <graph name>;

<graph type> := UNGRAPH | DIGRAPH



4. Run a Relational Query

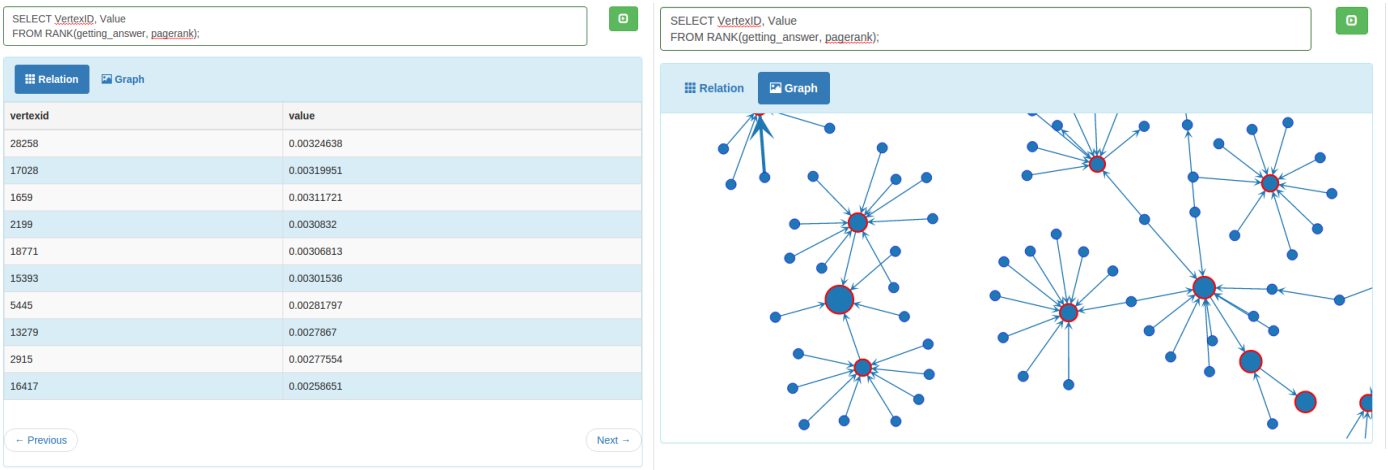
Rogas uses the Psycopy (a PostgreSQL adapter) to run the SQL operations, so most of the PostgreSQL operations can be processed well in Rogas.



5. Run a RANK Query

Syntax of the ranking operation:

RANK (<graph name>, <measure>)
<measure> := degree | indegree | outdegree | betweenness | closeness | pagerank

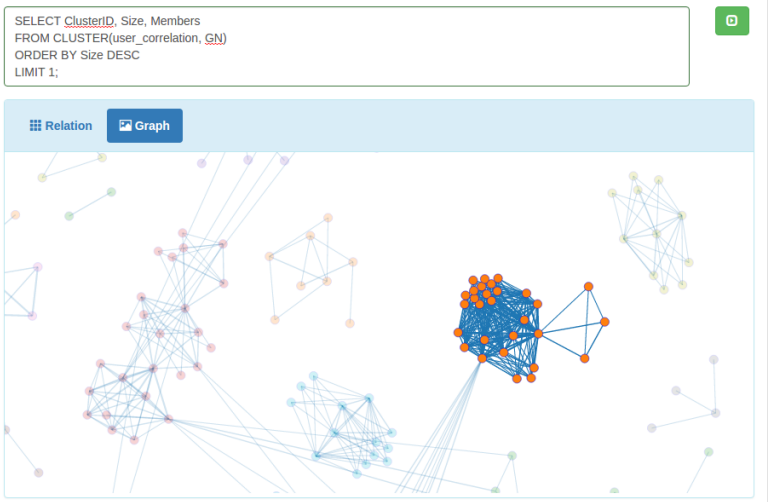


6. Run a CLUSTER Query

Syntax of the clustering operation:

CLUSTER(<graph name>, <algorithm>)
<algorithm>:=CC | SCC | GN | CNM | MC

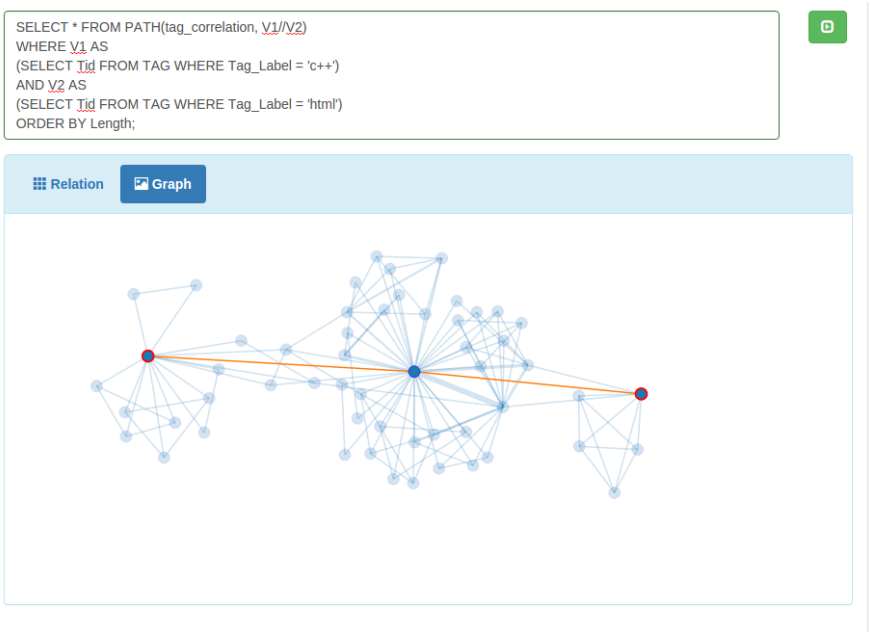
CC: finding connected components,
SCC: finding strongly connected components
GN: Girvan-Newman algorithm
CNM: Clauset-Newman-Moore algorithm
MC: Peixoto's modified Monte Carlo algorithm



7. Run a PATH Query
Syntax of the path finding operation:

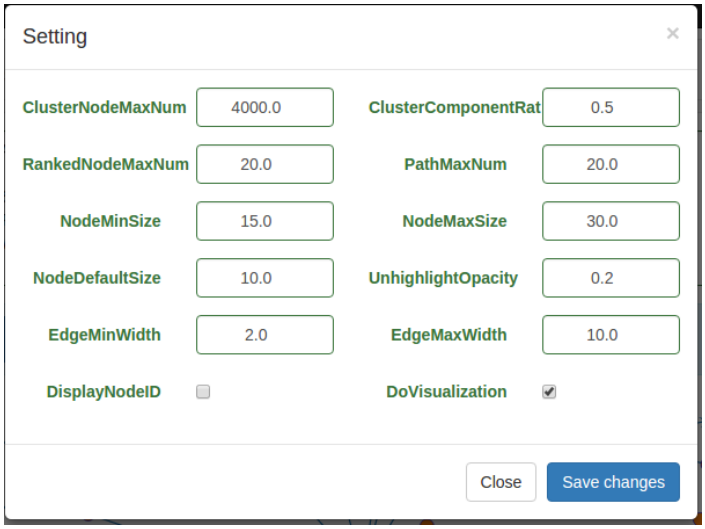
PATH(<graph name>, <path expression>)
<path expression> := . | V | <path expression>/<path expression> | <path expression>//<path expression>

V is a vertex expression that imposes certain condition on the vertices of a path, '.' is a do-not-care symbol indicating that any vertex is allowed in its position, '/' represents one edge, and '//' represents any number of edges. A path expression is valid if it contains a vertex expression in the first and last positions.



8. Setting

ClusterNodeMaxNum = 4000 means if the graph has more than 4000 nodes, the system will only show 4000 nodes among the graph, but the system will rescale the size of each cluster according to the proportion in order to maintain the structure of clusters.



ClusterComponentRat is the factor of rescaling. 0.5 means the system displays half nodes as selected nodes from the connected components and half nodes from the neighbor nodes around the selected nodes.

RankedNodeMaxNum 20 means if the user doesn't specify Top-k nodes in the ranking operation, the system will display Top-20 nodes by default.

PathMaxNum 20 means the system will display at most 20 neighbor nodes around the selected nodes.

NodeMinSize, NodeMaxSize and NodeDefaultSize control the size of selected nodes. For CLUSTER and PATH operations, all the nodes are displayed in NodeDefaultSize. For RANK operations, nodes are displayed according to their ranking values scaled from the NodeMinSize to the NodeMaxSize.

EdgeMinWidth and EdgeMaxWidth control the width of each edge. If two nodes have connection more than once, then the width of the edge will increase.

DisplayNodeID: Whether or not display the NodeID in the graph.

DoVisualization: If it is canceled, then the system will not show the "Graph" result tab. If you want to create a large graph, it is recommended to turn off the visualization function to save processing time.

Sample Queries of the StackOverflow dataset

The following queries are just trying to provide comprehensive samples for most types of queries supported by the system. Perhaps some queries are not very reasonable, for example, in the StackOverflow network, finding a path is not very useful. All the queries correspond to the “stack.bak”[2] sample dataset.

1. Python user: Users who have replied to questions that are labelled by "Python" tag

```
SELECT Owner_id FROM ANSWER
WHERE Parent_Qid IN
(
        SELECT Qid FROM LABELLED_BY
        WHERE Tid IN
        (
                SELECT Tid FROM TAG
                WHERE Tag_label = 'python'
        )
);
```

2. Influence of users (S_s1 USER <-> GETTING ANSWER)

```
CREATE DIGRAPH getting_answer AS
(
        SELECT q.Owner_id AS Q_User, a.Owner_id AS A_User
        FROM QUESTION as q, ANSWER as a
        WHERE q.Accepted_Aid = a.Aid AND q.Owner_id != a.Owner_id
);

SELECT VertexID, Value
FROM RANK(getting_answer, pagerank);
```

3.Python experts (Top k): Top 10 users who often reply python questions and their answers are often accepted. (S_s1 USER <-> GETTING ANSWER)

```
SELECT VertexID, Value
FROM RANK(getting_answer, pagerank)
WHERE VertexID IN
(
        SELECT Owner_id FROM ANSWER
        WHERE Parent_Qid IN
        (
                SELECT Qid FROM LABELLED_BY
                WHERE Tid IN
                (
                        SELECT Tid FROM TAG
                        WHERE Tag_label = 'python'
                )
        )
)
LIMIT 10;
```

[2] stack.bak is a small part of the StackOverflow data from http://snap.stanford.edu/proj/snap-icwsm/

4.Biggest community of correlation users: Find the biggest community of users who often reply same questions (they may have common interest) (S_s2 USER <-> User Correlation)

```
CREATE UNGRAPH user_correlation AS
(
        SELECT a1.Owner_id AS User, a2.Owner_id AS R_User
        FROM ANSWER AS a1, ANSWER AS a2
        WHERE a1.Parent_Qid = a2.Parent_Qid AND a1.Owner_id != a2.Owner_id
        LIMIT 1000
);

SELECT ClusterID, Size, Members
FROM CLUSTER(user_correlation, GN)
ORDER BY Size DESC
LIMIT 1;
```

5.Competitive users: users who are in the biggest community of correlative users and they also have big influence in the community (one of their answers may be accepted).
(S_s1 USER <-> GETTING ANSWER) AND (S_s2 USER <-> User Correlation)

```
SELECT r.VertexID
FROM RANK(getting_answer, pagerank) AS r,
(
        SELECT Members
        FROM CLUSTER(user_correlation, CNM)
        ORDER BY Size DESC
        LIMIT 1
) AS c
WHERE r.VertexID = ANY(c.Members);
```

6. Related Question:Find the correlation groups of questions which often use same tags (S_s3 QUESTION <-> QUESTION CORRELATION)

```
CREATE UNGRAPH question_correlation AS
(
        SELECT l1.Qid AS Qid, l2.Qid AS R_Qid
        FROM LABELLED_BY AS l1, LABELLED_BY AS l2
        WHERE l1.Tid = l2.Tid AND l1.Qid != l2.Qid
        LIMIT 1000
);

SELECT ClusterID, Size, Members
FROM CLUSTER(question_correlation, CNM)
ORDER BY Size DESC;
```

7. Related Tag:Find the correlation groups of tags which are often used on same questions (S_s4 TAG <-> TAG CORRELATION)

```
CREATE UNGRAPH tag_correlation AS
(
        SELECT l1.Tid AS Tid, l2.Tid AS R_Tid
        FROM LABELLED_BY AS l1, LABELLED_BY AS l2
        WHERE l1.Qid = l2.Qid AND l1.Tid != l2.Tid
        LIMIT 1000
);

SELECT ClusterID, Size, Members
FROM CLUSTER(tag_correlation, MC)
ORDER BY Size DESC;
```

8. Finding Path: see how the 'c++' tag connects with the 'html' tag

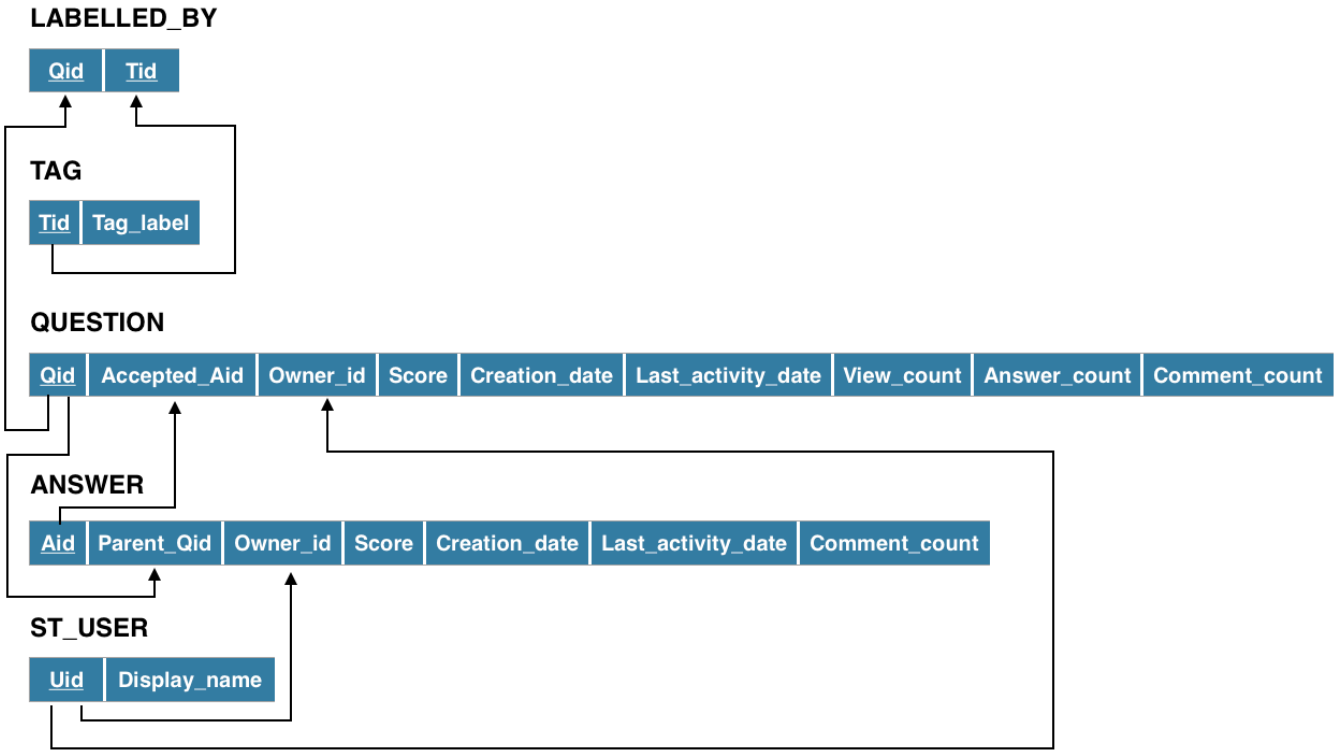
```
SELECT * FROM PATH(tag_correlation, V1//V2)
WHERE V1 AS
(SELECT Tid FROM TAG WHERE Tag_Label = 'c++')
AND V2 AS
(SELECT Tid FROM TAG WHERE Tag_Label = 'html')
ORDER BY Length;
```

Sample Schema of Three Real Networks

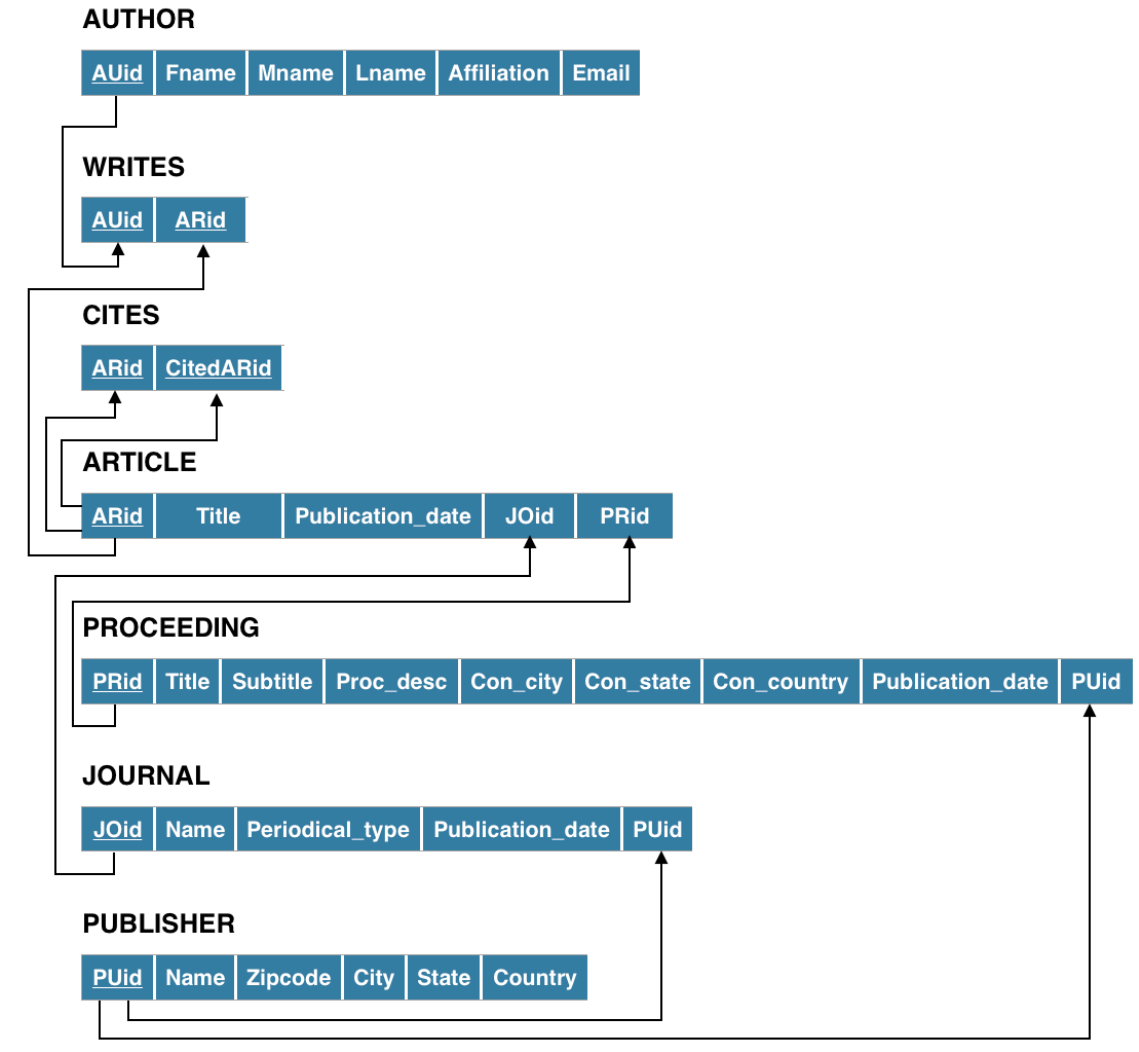
In this section, we list three sample database schema as references for importing your own dataset into the system. The database schema here is only an example and you certainly can design the schema as your own wish. Besides the database schema for the Stack Overflow network (our sample dataset), this section includes the database schema for a bibliographical network (e.g. ACM or DBLP network) and a social network (e.g. Twitter or Weibo).

The underlined attributes represent primary keys and each directed arc represents a foreign key.

1. Stack Overflow Network:



2. ACM Bibliographical Network:



3. Twitter Network: