gStore1.3 Version User Guide

1.Update Log

1.1 Code Version Update Log

gStore 1.3 Version

- Update Time：2024-9-9
- Update functions
    - **Directory Structure Refactoring**: Refactored the directory structure and optimized the compilation method, improving compilation efficiency by 30%.
    - **New Inference Engine Module**: Supports dynamic management and real-time invocation of inference rules, and supports property inference and relationship inference based on ontology models.
    - **New Built-in Advanced Functions**: Added four advanced functions: Diameter Estimation (diameterEstimation), Betweenness Centrality (betweennessCentrality), Jaccard Similarity (JaccardSimilarity), and Degree Correlation (degreeCorrelation).
    - **New Aggregate Function SAMPLE**: Supports calling in SELECT queries.
    - **New API Interfaces**: Added `checkOperationState` and `reasonManage` interfaces.
    - **Kernel Logging Refactor**: Supports configuration of five logging levels and expression-based logging content configuration.
    - **Optimization of CSR Resource Update Mechanism**: CSR resources are dynamically updated based on query needs after data updates.
    - **Cache Management Optimization**: Optimized the issue where system caches were not being actively released, leading to excessive server memory consumption.
    - **Data Update Logic Optimization**: Improved the logic for batch addition and deletion of data, increasing execution efficiency by more than 40%.
    - **Added Data Compression Mechanism**: The HTTP API now supports gzip compression, reducing data transfer size to within 20% of the original data, and shortening data transmission time.
    - **API Interface Optimization**: Added asynchronous mode and callback mode to some time-consuming interfaces.
    - **Local Command Optimization**: Optimized local commands, such as build, batch add, and delete, with support for zip file types.
    - **Bug Fixes**: Fixed a series of bugs.

gStore 1.2 Version

- Update Time：2023-11-11
- Update functions:
    - **Optimizing ORDER BY statements**: streamlining the execution logic of ORDER BY, removing unnecessary type judgments and conversions, and significantly improving

execution efficiency.
    - **Optimized Build Module**: Supports building empty libraries.
    - **Optimizing the Triple Parser**: Supports pure numeric IRIs, IRIs consisting only of numbers and letters, and IRIs starting with numbers.
    - **New API interfaces**: gStore 1.2's ghttp and gRPC services have added five interfaces for **uploading files**, **downloading files**, **counting system resources**, **renaming**, and **obtaining backup paths**.
    - **New built-in advanced functions**: gStore 1.2 version adds seven advanced functions, namely **single source shortest path (SSSP, SSSPLen)**, **label propagation (labelProp)**, **weakly connected component (WCC)**, **global/local clustering coefficient (clusteringCoeff)**, **louvain algorithm (louvain)**, **K-hop count (kHopCount)**, and **K-hop neighbor (kHopNeighbor)**.
      -* * Added support for calling CONCAT functions in SELECT statements * *.
    - **Optimizing some local commands and API interfaces**: Optimizing the local command gconsole, optimizing the interfaces for building, loading, and statistical graph databases, and fixing potential bugs that may lead to memory leaks.
    - **Support for Multiple Data Formats**: Added support for multiple formats such as **Turtle**, **TriG**, **RDF/XML**, **RDFa**, and **JSON-LD**.
    - **Optimization of custom graph analysis algorithm editing function**: Redesign the interface of the custom graph analysis algorithm editing function, optimize the dynamic compilation algorithm, and improve compilation efficiency.
    - **Bug fixes**: Fixed a series of bugs.

gStore 1.0 Version

- Update Time：2022-10-01
- Updated functions:
    - Support of user-defined graph analysis functions: users can manage their own graph analysis functions through the API interfaces or the visual management platform gStore-workbench. Users can obtain the number of nodes and edges of the graph and neighbors of any given node, etc. through interface functions and use them as basic units to implement their own graph analysis functions. Dynamic compilation and execution of user-defined graph analysis functions are supported.
    - The gRPC network interface service: gRPC is a high-performance network interface service based on HTTP protocol implemented based on the open source library `workflow`, which further improves the efficiency and stability of the interface service. Experiments show that gRPC achieves a great improvement in concurrent access performance compared with ghttp, the previous network interface; for example, in the case of **2000/QPS**, the rate of denied access is **0%**.
    - gConsole module: in gStore 1.0, we launched the gConsole module, which enables the long-session operation of gStore with contextual information.
    - Decoupling of the optimizer and executor: gStore 1.0 decouples the optimizer and executor, converting from the original deeply coupled greedy strategy to a query optimizer based on dynamic programming and a query executor based on breadth-first traversal.
    - Optimization of Top-K queries: We implemented a Top-K SPARQL processing framework based on the DP-B algorithm in gStore, including query segmentation and sub-result aggregation.
    - Support of ACID transactions: by introducing the multi-version management mechanism, gStore 1.0 can start ACID transactions for insert and delete operations, which users can open, commit, and roll back. Currently gStore 1.0 supports four isolation levels: read-uncommitted, read-committed, repeatable read and serializable.

- Reconstruction of database kernel and optimization of the plan tree generation logic: in gStore 1.0, two types of join operations (worst-case-optimal joins and binary joins) are introduced to optimize query execution and further improve query efficiency.
- Optimized logging module: based on the log4cplus library, the system logs can be output in a unified format. Users can configure the log output mode (console output or file output), output format, and output level.
- New built-in advanced functions: gStore 1.0 supports four new advanced functions, namely triangleCounting, closenessCentrality, bfsCount and kHopEnumeratePath.
- Extended support for BIND statements: gStore 1.0 supports assigning values to variables using algebraic or logical expressions in BIND statements.
- Optimization of some local commands and API interfaces (e.g., the shutdown command), and fixing a series of bugs (e.g., more accurate gmonitor statistics).

gStore 0.9.1 Version

- Update Time：2021-11-23
- Updated functions:
  - By separating gStore kernel parsing and execution, the query performance can be further improved through join order and other technologies, and the performance can be improved by more than 40% in complex queries;
  - Rewrite the http service component ghttp of gStore, and add user permissions, heartbeat detection, batch import, batch delete and other functions, and write a standard GHTTP API document (see interface list), further enrich the functions of GHTTP, improve the robustness of GHTTP;
  - Add Personalized PageRank (PPR) custom function, which can be used to calculate the relatedness between entities to find the node with the most influence;
  - Added support for arithmetic and logical operations in Filter statements, such as arithmetic operations (e.g.? x + ? Y = 5); Logical operations (e.g.? x + ? y = 5 && ? Y > 0);
  - The gServer component is added to realize two-way Socket API communication. Users can access gStore remotely through GHTTP component and gServer component;
  - The local operation command format is planned, and the --help command is introduced. Users can view the detailed command format of each function. For example, bin/gbuild -h/--help can view the detailed command format of gbuild;
  - Fixed a number of bugs.

gStore 0.9.0 Version

- Update Time：2021-02-10
- Updated functions：
  - Upgrade the SPARQL parser generator from ANTLR V3 to the latest, well-documented, and well-maintained V4;
  - Support writing numeric literals without data type suffixes in SPARQL queries;
  - Support for arithmetic and logical operators in SELECT clauses;
  - Supports aggregations of SUM, AVG, MIN, and MAX in the SELECT clause;
  - Additional support is built into filters, and function functions including `datatype`, `contains`, `ucase`, `lcase`, `strstarts`, `now`, `year`, `month`, `day`, and `abs`;
  - Supports path-related functions as an extension of SPARQL 1.1, including loop detection, shortest paths, and K-Hop reachability;
  - Supports full and incremental database backup and recovery. Administrators can enable automatic full backup;
  - Supports Log-based rollback operations;

    - Supports transactions with three levels of isolation: committed reads, snapshot isolation, and serializable ；
    - Expand the data structure to accommodate large-scale graphs of up to 5 billion triples.

1.2 Document Version Update Log

2024.8.30　gStore 1.3

- Modified the contents about Qucik Start to match gStore 1.3 version
- Modified the contents about Installation Instructions to match gStore 1.3 version
- Modified common API content to match gStore 1.3 version
- Modified SPARQL query language content to match gStore 1.3 version
- Added update log to record the version of gStore and related document updates

2023.11.11 gStore 1.2 Download

- gStore 1.2 version document is available for download
- Modified the contents about Qucik Start to match gStore 1.2 version
- Modified the contents about Installation Instructions to match gStore 1.2 version
- Modified common API content to match gStore 1.2 version
- Modified Workbench console content to match gStore 1.2 version
- Added gStore 1.2 version release in gStore Chronology
- Added update log to record the version of gStore and related document updates

2022.10.1 gStore 1.0 Download

- gStore 1.0 version document is available for download
- Modified the contents about Qucik Start to match gStore 1.0 version
- Modified the contents about Installation Instructions to match gStore 1.0 version
- Modified common API content to match gStore 1.0 version
- Modified Workbench console content to match gStore 1.0 version
- Added gStore 1.0 version release in gStore Chronology
- Added update log to record the version of gStore and related document updates

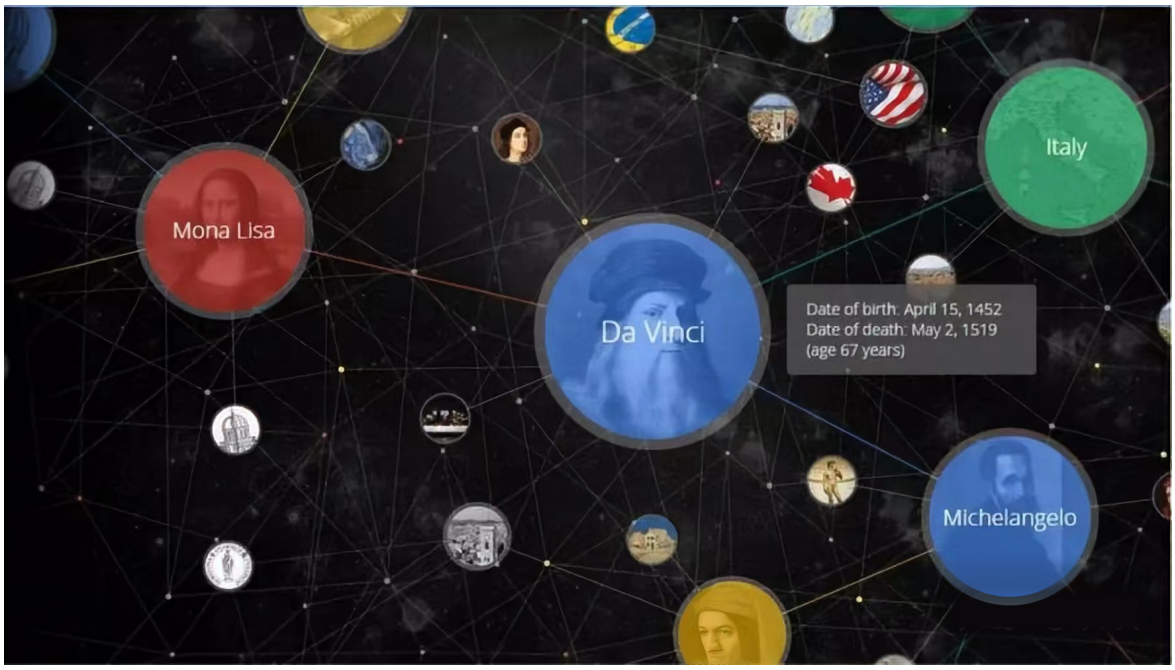
Previously:

- Modified the contents about Quick Start to match gStore 0.9.1 version
- Modified common API content to match gStore 0.9.1 version
- Modified Workbench console content to match gStore 0.9.1 version
- Added update log to record the version of graph database gStore and related document updates
- Added a document download directory for users to download documents

2. Introduction of Knowledge Graph and gStore

2.1 Introduction of Knowledge Graph

In recent years, with the revival of the concept of "artificial intelligence", in addition to the hot term "deep learning", "knowledge graph" is undoubtedly another "silver bullet" in the eyes of researchers, industry and investors. To put it simply, "knowledge Graph" is a data model that displays "entities", "attributes" and "relationships" among entities in the form of Graph. Below is an example from Google's knowledge Graph introduction page. There are four entities in the example, "Da Vinci", "Italian", "Monlarissa" and "Michelangelo". This diagram clearly shows the individual attributes and attribute values of "Da Vinci" (e.g., name, date of birth, time of death, etc.) as well as the relationships between them (e.g., Monlarissa is a painting by Da Vinci, Da Vinci was born in Italy, etc.).



At present, the knowledge graph generally uses RDF(Resource Description Framework) model in semantic Web Framework to represent data. The Semantic Web is a concept proposed by Tim Berners-Lee, the father of the World Wide Web, in 1998. Its core is to build a data-centric network, namely the Web of Data. RDF is the standard for data description in the SEMANTIC Web framework of W3C. It is often called RDF triples (Subject, predicate, object). Where the principal must be a described resource, represented by a URI. A predicate can represent an attribute of a subject or a relationship between a subject and an object. When representing an attribute, the object is the attribute value, usually a literal. Otherwise the object is another resource represented by a URI. The figure below shows a knowledge graph dataset for RDF triples of a people encyclopedia. For example: y Abraham_Lincoln said an entity URI prefix (y = http://en.wikipedia.org/wiki/), it has three properties (hasName, BornOdate DiedOnDate) and a relationship (DiedIn).

Prefix: y= http://en.wikipedia.org/wiki/

Subject	Predict	Object
y:Abraham_Lincoln	hasName	“Abraham Lincoln”
y:Abraham_Lincoln	BornOnDate	“1809-02-12”
y:Abraham_Lincoln	DiedOnDate	1865-04-15
y:Abraham_Lincoln	DiedIn	y:Washington_D.C
y:Washington_D.C	hasName	“Washington D.C.”
y:Washington_D.C	FoundYear	1790
y:Washington_D.C	rdf:type	y:city
y:United_States	hasName	“United States”
y:United_States	hasCapital	y:Washington_D.C
y:United_States	rdf:type	Country
y:Reese_Witherspoon	rdf:type	y:Actor
y:Reese_Witherspoon	BornOnDate	“1976-03-22”
y:Reese_Witherspoon	BornIn	y:New_Orleans,_Louisiana
y:Reese_Witherspoon	hasName	“ReeseWitherspoon”
y:New_Orleans,_Louisiana	FoundYear	1718
y:New_Orleans,_Louisiana	rdf:type	y:city
y:New_Orleans,_Louisiana	locatedIn	y:United_States

Figure 1-1 RDF data example

For RDF data set, W3C proposes a structured query language SPARQL. It is similar to SQL, the query language for relational databases. Like SQL, SPARQL is also a descriptive structured query language. That is, users only need to describe the information they want to query according to the syntax rules defined by SPARQL, without specifying the steps of the computer to perform the query. SPARQL became an official W3C standard in January 2008. The WHERE clause in SPARQL defines the query criteria, which are also represented by triples. We do not cover much syntax detail, but interested readers may refer to [1]. The following example illustrates the SPARQL language. Suppose we need to query the RDF data above for "the name of a person born on February 12, 1809 and who died on April 15, 1865?" This query can be represented as a SPARQL statement as shown below. The figure below shows a knowledge graph dataset for RDF triples of a people encyclopedia. For example: y Abraham_Lincoln said an entity URI prefix (y = http://en.wikipedia.org/wiki/), it has three properties (hasName, BornOdate DiedOnDate) and a relationship (DiedIn).

Figure 1-2 SPARQL Query example

A core problem of knowledge graph data management is how to efficiently store RDF datasets and quickly answer SPARQL queries. In general, there are two completely different sets of thinking. First, we can use existing mature database management systems (such as relational database systems) to store knowledge graph data. SPARQL queries oriented to RDF knowledge

graph can be converted into queries oriented to such mature database management systems, such as SQL queries oriented to relational databases. Use existing relational database products or related technologies to answer queries. The core research problem is how to build relational tables to store RDF knowledge graph data and make the transformed SQL query statement query performance higher. The other is to directly develop a knowledge graph data storage and query system (Native RDF graph database system) for RDF knowledge graph data. Considering the characteristics of RDF knowledge graph management, optimization is carried out from the bottom of the database system.

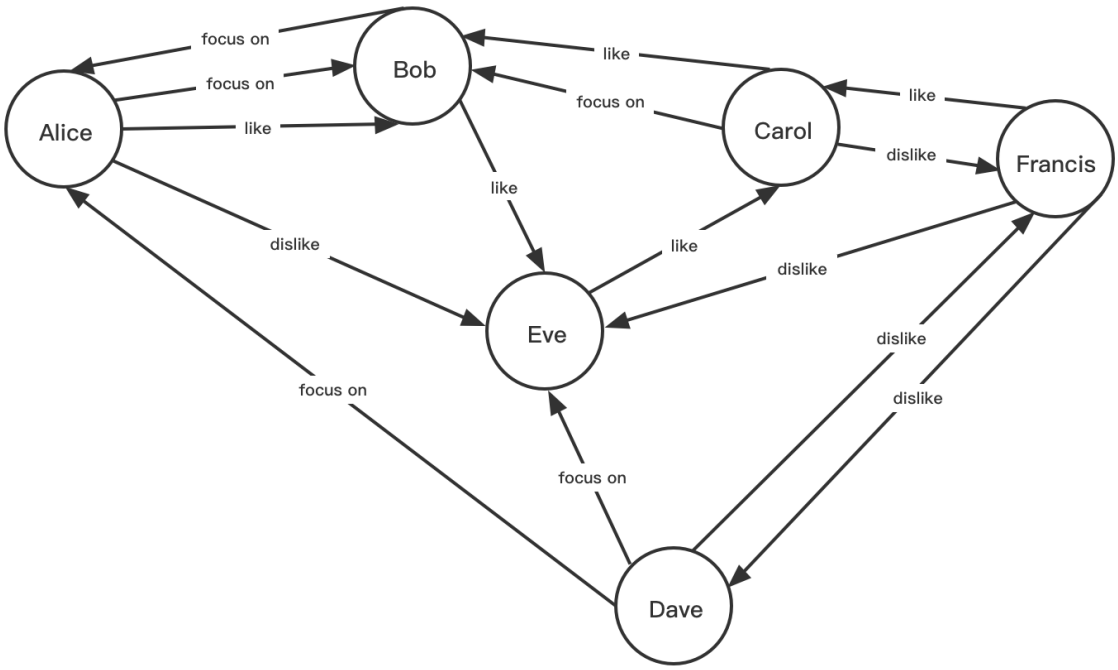


Figure 1-3 RDF graph and SPARQL query graph

2.2 Introduction of gStore

The gStore system developed by us belongs to the latter. gStore is an open source graph database system (commonly known as Triple Store) for RDF data model developed by data Management Laboratory (PKUMOD) of Wangxuan Institute of Computer Technology of Peking University after ten years. Different from traditional relational database based knowledge graph data management method , gStore **Native Graph Model**, maintains the **Graph structure of the original RDF knowledge graph**; Its data model is labeled, directed polygon graph, each vertex corresponds to a subject or object. We convert SPARQL queries for RDF to ** **Subgraph matching queries for RDF graphs**, the graph structure-based index (VS-tree) proposed by us is used to speed up query performance. Figure 1-3 shows the structure of the RDF graph and SPARQL query graph corresponding to the above example. Answering SPARQL queries is essentially finding the matching position of the subgraph of a SPARQL query graph in an RDF graph, which is the theoretical basis of answering SPARQL queries based on graph databases. In the example in Figure 1-3, the subgraph derived from nodes 005,009,010 and 011 is a match of the query graph, from which it is easy to know that the SPARQL query result is "Abraham Lincoln." For the core academic ideas of gStore, please refer to the published papers of Development Resources - Papers and Patents.

gStore begin withdata Management Laboratory (PKUMOD) of Wangxuan Institute of Computer Technology of Peking University (Lei Zou, Jinghui Mo, Lei Chen,M. Tamer Ozsu, Dongyan Zhao, gStore: Answering SPARQL Queries Via Subgraph Matching, Proc. VLDB 4(8): 482-493, 2011), VLDB 2011 paper by Prof. Lei Zou, Prof. Tamer Ozsu, University of Waterloo, and Prof. Lei Chen, Hong Kong University of Science and Technology , proposes a query execution scheme using subgraph matching to answer Basic Graph Pattern (BGP) statements in SPARQL. Since the publication of this paper, PKUMOD Laboratory has been continuously engaged in the open source, maintenance and system optimization of gStore system under the funding of the National Natural Science Foundation of China and the key RESEARCH and development projects of the Ministry of Science and Technology of China. At present, the open source gStore system on Github can support SPARQL 1.1 standard defined by W3C (see SPARQL Query Language for details).

After a series of tests, the results showed that gStore was faster than other database systems at answering complex queries (for example, containing circles). For simple queries, gStore and other database systems work fine. The standalone version of gStore can support more than **5 billion** RDF triples and SPARQL queries. The distributed system gStore (distributed version, not open source at present) has very good scalability. According to the test report given by "China Software Evaluation Center", The distributed gStore system has second query times on ten billion RDF triplet datasets.

Since the gStore system was opened on Github, BSD 3-clause, which is widely used in the open source community, has been adopted to promote the construction of gStore related knowledge graph technology ecology. According to this agreement, we require users to allow users to modify and redistribute codes freely on the premise of fully respecting the copyright of code authors, and also allow users to develop, distribute and sell commercial software freely on the basis of gStore codes. However, the above conditions must meet the relevant legal provisions stipulated in Chapter 10 "Legal Provisions" according to the BSD 3-clause open source agreement. We strictly require users to mark "powered by gStore" and the gStore logo (see gStore Logo for details) on the software they distribute based on the gStore code. We strongly recommend that users refer to the "Open Source and Legal Provisions" before using gStore.

3. Installation Instructions

3.1 System Requirements

Project	Requirement
Operating System	Linux, such as CentOS, Ubuntu etc.
Framework	x86_64, amd64, arm64, aarch64, loongarch64
Disk Size	Depends on the size of the data set
Memory Size	Depends on the size of the data set
unzip/bzip2	Must install
glibc	Must install version >= 2.14
gcc	Must install , recommend version >= 9.3.0
g++	Must install , recommend version >= 9.3.0
make	Must install
cmake	Must install version >= 3.23.2
pkg-config	Must install
uuid-devel	Must install
boost	Must install , recommend version >= 1.74
readline-devel	Must install
curl-devel	Must install
openssl-devel	Must install version >= 1.1
jemalloc	1.2 version and above, must be installed
openjdk-devel	If using the Java API, must install version = 1.8.x
requests	If using the Python http API, must install
node	If using the Node.js http API, must install version >=10.9.0
pthreads	If using the PHP http API, must install
ccache	Optional, used to speed up compilation

3.2 Compilation and Installation

3.2.1 Installation Environment

Running the corresponding scripts in scripts/setup/ for your operating system will automatically solve most of your problems for you(recommend). For example, if you are an Ubuntu user, you can execute the following command:

```
$ . scripts/setup/setup_ubuntu.sh
```

If you are a Centos user,you can execute the following command:

```
$ . scripts/setup/setup_centos.sh
```

Before running the script, we recommend you install gcc and g++ 5.0 or later.

Of course, you can also choose to manually prepare the environment step-by-step; Detailed installation instructions for each system requirement are provided below.

```
$ sudo -i
Please input your password
[sudo] xxx's password:
```

unzip/bzip2 installation

Used for extracting the gStore zip and bz2 packages during offline installation

Determine if unzip is installed

```
$ yum list installed | grep unzip       #centos系统
$ dpkg -s unzip                         #ubuntu系统
```

If not installed ,install

```
$ yum install -y unzip              #centos系统
$ apt-get install -y unzip         #ubuntu系统
```

Change the unzip in the above command to bzip2 can install bzip2

gcc and g++ installation

Check g++ version:

```
$ g++ --version
```

Using 9.3.0 as an example :(for Ubuntu and CentOS)

```
$ cd /var/local/
$ mkdir gcc
$ cd gcc
$ wget https://mirror.linux-ia64.org/gnu/gcc/releases/gcc-9.3.0/gcc-9.3.0.tar.gz
2>&1
$ tar -zxf gcc-9.3.0.tar.gz
$ yum install -y gcc-c++ unzip bzip2 ntpdate m4
$ ntpdate -u ntp.api.bz
$ cd gcc-9.3.0
$ ./contrib/download_prerequisites
$ln -sf gmp-6.1.0 gmp
```

```
$ ln -sf mpfr-3.1.4 mpfr
$ ln -sf mpc-1.0.3 mpc
$ mkdir gcc-make-tmp
$ cd gcc-make-tmp
$ ../configure --prefix=/opt/gcc/9.3.0 --enable-checking=release --enable-
languages=c,c++ --disable-multilib 2>&1
$ make -j4 2>&1
$ make install 2>&1
$ echo "/opt/gcc/9.3.0/lib64" >> /etc/ld.so.conf.d/libgcc-x86_64.conf
$ mv /opt/gcc/9.3.0/lib64/libstdc++.so.6.0.28-gdb.py /opt/gcc/9.3.0
$ echo 'export PATH=$PATH:'"/opt/gcc/9.3.0/bin" >> /etc/profile
$ yum remove -y gcc
$ yum remove -y gcc-c++
$ cd ../../
$ rm -rf gcc-9.3.0
$ ldconfig
$ source /etc/profile
```

Ubuntu can also be installed directly using the following commands:

```
$ apt install -y gcc-9 g++-9
```

readline-devel installation

Check whether readline-devel is installed

```
$ yum list installed | grep readline-devel  # CentOS
$ dpkg -s readline                          # Ubuntu
```

if not, install it

```
$ sudo yum install readline-devel       # CentOS
$ sudo apt install -y libreadline-dev   # Ubuntu
```

pkg-config installation

Check whether pkg-config is installed

```
$ pkg-config --version      #centos
$ pkg-config --version      #ubuntu
```

if not , install it

```
$ yum install -y pkgconfig.x86_64      #centos
$ apt install -y pkg-config            #ubuntu
```

uuid-devel installation

Check whether uuid-devel is installed

```
$ yum list installed | grep libuuid-devel    #centos
$ dpkg -s uuid-dev                           #ubuntu
```

if not, install it

```
$ yum install -y libuuid-devel      #centos
$ apt install -y uuid-dev           #ubuntu
```

boost installation

Recommend using version 1.74.0 to install

Check whether boost is installed

```
$ yum list installed | grep boost       # CentOS
$ dpkg -s boost                         # Ubuntu
```

If not, install it:　(use version 1.74.0 as example)

version:1.74.0

address: http://sourceforge.net/projects/boost/files/boost/1.74.0/boost_1_74_0.tar.gz

Installation script: (for CentOS and Ubuntu)

```
$ wget
http://sourceforge.net/projects/boost/files/boost/1.74.0/boost_1_74_0.tar.gz
$ tar -xzvf boost_1_74_0.tar.gz
$ cd boost_1_74_0
$ ./bootstrap.sh --with-libraries=system,regex,thread,filesystem
$ ./b2 -j4
$ ./b2 install
```

Ubuntu can also be installed directly using the following commands:

```
$ sudo apt install -y libboost-all-dev
```

Note: please install boost after ensuring that the g++ version is above 9.3.0 Undefined reference to 'boost::...' - undefined reference to 'boost::... '"), most likely because you compiled Boost with GCC versions lower than 9.3.0. At this point, recompile Boost using the following step:

- Clear old files: `./b2 --clean-all`
- In the user-config.jam file under./tools/build/ SRC (if this file does not exist under this path, Please find a sample user-config.jam file under./tools/build/example or some other directory and copy it to./tools/build/ SRC) to add ： `using gcc : 9.3.0 : gcc-9.3.0's path ;`
- Run under ./ `./bootstrap.sh --with-toolset=gcc`
- `sudo ./b2 install --with-toolset=gcc`

Then recompile gStore (please start from 'make pre')

```
# change directory
$ cd build
# clean 3rd lib
$ make clean_pre
# precompile again
$ make pre
# compile
$ make -j4
# init database
$ make init
```

curl-devel installation

Check whether curl-devel is installed

```
$ yum list installed | grep readline-devel  # CentOS
$ dpkg -s readline                          # Ubuntu
```

if not, then install:

version: 7.55.1

address: https://curl.haxx.se/download/curl-7.55.1.tar.gz

Installation scripts (for CentOS and Ubuntu)

```
$ wget https://curl.haxx.se/download/curl-7.55.1.tar.gz
$ tar -xzvf  curl-7.55.1.tar.gz
$ cd curl-7.55.1
$ ./configure
$ make
$ make install
```

Or use the following command to install

```
$ yum install -y curl libcurl-devel              # CentOS
$ apt-get install -y curl libcurl4-openssl-dev   # Ubuntu
```

openssl-devel installation

Check whether openssl-devel is installed

```
$ yum list installed | grep openssl-devel    # CentOS
$ dpkg -s libssl-dev                         # Ubuntu
```

If not installed, install it

```
$ yum install -y openssl-devel                # CentOS
$ apt-get install -y libssl-dev zlib1g-dev    # Ubuntu
```

cmake installation

Check whether cmake is installed

```
$ cmake --version               # CentOS
$ cmake --version               # Ubuntu
```

if not, install:

version: 3.23.2

address: https://curl.haxx.se/download/curl-7.55.1.tar.gz

Installation scripts (for CentOS and Ubuntu)

```
$ wget https://cmake.org/files/v3.23/cmake-3.23.2.tar.gz
$ tar -xvf cmake-3.23.2.tar.gz && cd cmake-3.23.2/
$ ./bootstrap
$ make -j4
$ make install
```

Ubuntu can also be installed directly using the following commands:

```
$ sudo apt install -y cmake
```

jemalloc installation

Determine whether jemalloc is installed

```
$ yum list installed | grep jemalloc        #Centos
$ dpkg -s libjemalloc-dev                 #Ubuntu
```

If not installed, install it

```
$ yum install -y jemalloc                   #Centos
$ apt-get install -y libjemalloc-dev    #Ubuntu
```

create soft link

```
# Locate the dynamic library, the default installation path is usually
/usr/lib64
$ find / -name libjemalloc*
/usr/lib64/libjemalloc.so.1

# if /usr/lib64/libjemalloc.so not find, create soft link
$ cd /usr/lib64
$ ln -s libjemalloc.so.1 libjemalloc.so
```

3.2.2 gStore obtainment

If you encounter permission issues, please prefix the command with `sudo` .

Method 1: download

gStore has been uploaded to gitee (code cloud), which is recommended for faster download for users in mainland China. The website is https://gitee.com/PKUMOD/gStore.
You can also open https://github.com/pkumod/gStore, download gStore.zip, then decompress the zip package.

Method 2: clone (Recommended)

Run the following code to clone:

```
$ git clone https://gitee.com/PKUMOD/gStore.git  # gitee (code cloud) faster for users in mainland China
$ git clone https://github.com/pkumod/gStore.git # GitHub

# using the -b parameter when getting historical versions, such as -b 1.2
$ git clone -b 1.2 https://gitee.com/PKUMOD/gStore.git
$ git clone -b 1.2 https://github.com/pkumod/gStore.git
```

Note: Git need to be installed first.

```
$ sudo yum install git      # CentOS
$ sudo apt-get install git  # Ubuntu
```

3.2.3 gStore compilation

Switch to the gStore directory:

```
$ cd gstore
```

There are two methods to compile gstore , compile by script (recommend) and compile manually

compile by script(recommend)

```
$ bash scripts/build.sh
#若编译顺利完成，最后会出现 Compilation ends successfully! 结果
```

compile manually

```
#change dirctory
$ cd gStore
# create build dirctory
$ mkdir build
$ cd build
# create the makefile  and other files
$ cmake ..
# precompile
$ make pre
# compile
$ make -j4
# init databases
```

```
$ make init
# the following result shows compilation and initalzation successful
---Compilation ends successfully!
---system.db is built successfully!
```

3.3 Deploy gStore using Docker

> We provide two ways to deploy gStore from containers:
>
> One is to build it yourself from the Dockerfile file in the project root and then run the container.
>
> Another option is to download the image that has been automatically built and run it directly.

3.3.1 environment preparation

Docker, refer to the address docker

3.3.2 Run by pulling the image directly(recommend)

Currently, the gstore image has been built and published on Docker Hub. The image is based on Ubuntu 16.04 and can be installed and deployed using the following methods.

(1) Pull the docker image

```
docker pull pkumodlab/gstore-docker:latest #Pull up the latest docker image
```

(2) Run the image

View the list of Docker images with the following command

```
docker image list
```

As shown in the following figure:

```
[root@iZ8vb3et5bz7170h9ealj1Z ~]# docker image list
REPOSITORY                          TAG                 IMAGE ID            CREATED
docker.io/pkumodlab/gstore-docker   latest              9ca4388fc81e        About an
minusdust/dice                      latest              df4ee1f016f5        8 weeks a
docker.io/pkumodlab/gstore-docker   <none>              6ba12fe64a1d        3 months
```

Start the image using the following command

```
docker run -itd -p 9999:9000 9ca4388fc81e /bin/bash
# Map container 9000 port to host 9999 port
# 9ca4388fc81e is image id
```

After the startup is complete, enter the following command to view container information

```
docker ps
```

```
[root@iZ8vb3et5bz7170h9ealj1Z ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
a5016bd46094        9ca4388fc81e        "/bin/bash"         2 minutes ago       Up 2 mi
```

Use the following command to enter the container

```
docker exec -it a5016bd46094 /bin/bash
#It is recommended to use exec to enter the container, and the container will
not stop after exiting
```

After entering the container, you can see the gstore directory in the root directory, which has already been compiled and installed.

You can operate it like using local gstore. To exit the docker container, you can use the following command:

```
exit
```

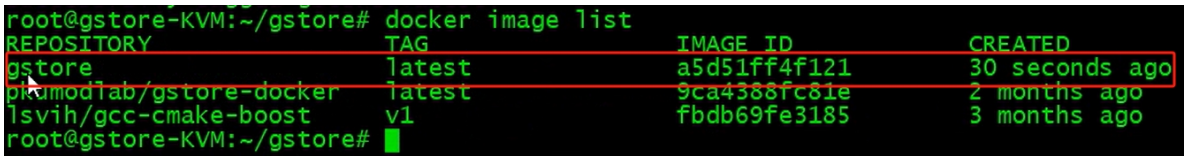
3.3.3 Build and run image locally

Build an image named gstore

```
# build docker image
build -t gstore

# show docker image
docker image list
```

As shown below



Start the locally built image

```
# Map host 9999 port to container 9000 port(Default port for the gstore API
service)
# a5d51ff4f121 is image id
docker run -itd -p 9999:9000 a5d51ff4f121 /bin/bash

# Show container status
docker ps -a
#CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS
      PORTS                                       NAMES
#317e4ce0a667   a5d51ff4f121   "bash /docker-entryp…"   6 minutes ago   Up 6
minutes   0.0.0.0:9999->9000/tcp, :::9999->9000/tcp   awesome_greider
```

Check API Service

```
curl http://127.0.0.1:9999 -X POST -H 'Content-Type: application/json' -d
'{"operation":"check"}'

# If the following information is displayed on the console, the gstore http api
service starts normally
# {"StatusCode":0,"StatusMsg":"the ghttp server is running..."}
```

There are probably a lot of other things that need to be added, so for now I've just added a basic version. The basic environment build is just the first step in containerization.

3.4 Deploy gStore Using a Static Installation Package

3.4.1 Static Package Download Links

We now offer four types of static installation packages for download and installation:

```
http://file.gstore.cn/f/eb7fa4c9d0d3421695f7/?dl=1   #linux-arm
http://file.gstore.cn/f/bb59558af0be44a6970d/?dl=1   #linux-x86
http://file.gstore.cn/f/158654beb04343ba9afe/?dl=1   #ubuntu-arm_64
http://file.gstore.cn/f/db5a9ac955ea45bfba27/?dl=1   #ubuntu-x86_64
```

3.4.2 Download static installation package

```
$ wget --content-disposition http://file.gstore.cn/f/75643cb217604efca75b/?dl=1
 #the address of the static package
$ tar -zxvf gstore-1.3-static-linux-arm.tar.gz
```

3.4.3 Initialize system

```
$ bin/ginit
```

3.5 Version upgrade

3.5.1 Method 1

Older versions are deployed as an upgrade by clone

Preparation before upgrade

```
# Go to the directory where gstore is deployed
$ cd gstore
# Pull the update script
$ git pull
# Check whether the version_checkout.sh file exists
$ ls scripts/version_checkout.sh
# If not, specify a branch fetch, such as 1.2
$ git status
1.2
$ git pull origin 1.2
# Check whether the upgrade version exists on the remote device, such as 1.3
$ git branch -r | grep origin/1.3
# Check whether there are conflicting branch names in the local area, such as
1.3(if yes, back up the branch and delete the conflicting branch)
$ git branch | grep 1.3
```

upgrade

```
# Go to the directory where gstore is deployed
$ cd gstore
# Specify the upgrade version and execute the script to upgrade, such as1.3
$ bash scripts/version_checkout.sh 1.3
# To choose whether to overwrite the database, enter y or n
the new version already has a xxx.db, overwrite [Y/n]
# Upgrade successfully
version update successfully .
```

Important data will be backed up. File directory: version+Version number+Date

3.5.2 Method 2

Older versions are deployed by downloading zip packages or static packages

```
# Get the old version gstore path
$ pwd
/xxx/xxxx/gstore
# Unzip the update package and go to the gstore directory
$ cd gstore/
# Specify the path of the old version and execute the script to upgrade
$ bash scripts/version_update.sh /xxx/xxxx/gstore
# To choose whether to overwrite the database, enter y or n
the new version already has a xxx.db, overwrite [Y/n]
# Upgrade successfully
version update successfully .
```

4. Quick Start

4.1 Data Format

```
`gStore` is a graph database engine based on the RDF model, and its data format follows the RDF model. RDF, the W3C standard for describing real-world resources, is a general way to describe information so that it can be read and understood by computer applications. Any entity in the real world can be represented as a resource in the RDF model, such as a book's title, author, modification date, content, and copyright information. These resources can be used to abstract concepts, entities, and events from the objective world in the knowledge graph. An attribute of each resource and its attribute value, or its relationship to other resources, is called a piece of knowledge. Properties and relationships can be represented as triples.

    A triad consists of three elements: Subject, Property 1, and Object. It usually describes the relationship between two resources or some Property of a resource. When a triple describes the attributes of a resource, its three elements are also called body, attribute, and Property Value. For example, the triad "Aristotle, Birthplace, Chalcis" expresses the fact that Aristotle was born in Chalcis.

   Using these properties and relationships, a large number of resources can be linked together to form a large RDF knowledge graph dataset. Therefore, a knowledge graph can often be viewed as a collection of triples. These collections of triples, in turn, form an RDF dataset. The triplet set of knowledge graph can be stored in relational database or graph database.gStore 1.2 adds support for various RDF serialization formats such as **Turtle**, **TriG**, **RDF/XML**, **RDFa**, and **JSON-LD**. Here are some of the mainstream data formats.
```

4.1.1 N-Triple

```
  RDF data should be provided in n-triple format (XML is not currently supported), and queries must be provided in SPARQL1.1 syntax. The following is an example of the n-triple format file:
```

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
_:a  foaf:name   "Johnny Lee Outlaw" .
_:a  foaf:mbox   <mailto:jlow@example.com> .
_:b  foaf:name   "Peter Goodguy" .
_:b  foaf:mbox   <mailto:peter@example.org> .
_:c  foaf:mbox   <mailto:carol@example.org> .
```

Triples are typically stored in the W3C-defined NT file format and represent three RDF data, where the values wrapped in `<` and `>`are urls of an entity, and the values wrapped in '"" are literals representing the value of an attribute of the entity, followed by'`^^`to indicate the type of the value. The following three RDF data points represent two attributes of `John`, `gender` and `age`, with values of `male` and `28` respectively. The last one indicates that `John` and `Li` have a `friend` relationship.

```
<John> <gender> "male"^^<http://www.w3.org/2001/XMLSchema#String>.
<John> <age> "28"^^<http://www.w3.org/2001/XMLSchema#Int>.
<John> <friend> <Li>.
```

```
More specific information about N-Triple please check [N-Triple]
(https://www.w3.org/TR/n-triples/). Not all syntax in SPARQL1.1 is parsed and
answered in gStore; for example, property paths are beyond the capabilities of
the gStore system.
```

4.1.2 Turtle

Turtle is a more concise serialization format for RDF. It enhances readability and conciseness on the basis of N-triple, and adds abbreviation functions such as prefixes. The following abbreviation form can represent four RDF three-month meetings, where multiple predicates of the same subject can be separated by `;` and multiple predicates of the same subject and predicate can be separated by `,`. The example file of Turtle is as follows:

```
@prefix swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
@prefix dbpedia: <http://dbpedia.org/resource>.
@prefix dbpedia-owl: <http://dbpedia.org/ontology>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

swp:BaronWayAppartment swp:hasNumberOfBedrooms "3"^^xsd:integer;
                       swp:isPartOf swp:BaronWayBuilding.

swp:BaronWayBuilding dbpedia-owl:location dbpedia:Amsterdam,
                                          dbpedia:Netherlands.
```

```
Turtle is usually stored in the ttl file format defined by the W3C. For a more
detailed description of the Turtle file, please refer to [Turtle]
(https://www.w3.org/TR/turtle/)。
```

4.1.3 TriG

TriG is an extension of the Turtle language that allows further naming of graphs, as shown in the following example：

```
# This document contains a same data as the
previous example.

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

# default graph - no {} used.
<http://example.org/bob> dc:publisher "Bob" .
<http://example.org/alice> dc:publisher "Alice" .

# GRAPH keyword to highlight a named graph
# Abbreviation of triples using ;
GRAPH <http://example.org/bob>
{
   [] foaf:name "Bob" ;
      foaf:mbox <mailto:bob@oldcorp.example.org> ;
      foaf:knows _:b .
}

GRAPH <http://example.org/alice>
{
```

```
    _:b foaf:name "Alice" ;
        foaf:mbox <mailto:alice@work.example.org>
}
```

4.1.4 RDF/XML

RDF/XML is a syntax defined by the W3C for expressing (i.e., serializing) RDF graphs as XML documents. RDF/XML is sometimes misleadingly referred to as RDF because it was introduced in other W3C specifications that define RDF, and historically, it was the first W3C standard RDF serialization format. For a more detailed description of RDF/XML files, please refer to **RDF/XML**.The following is an example::

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/">

  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
    <dc:title>RDF 1.1 XML Syntax</dc:title>
    <dc:title xml:lang="en">RDF 1.1 XML Syntax</dc:title>
    <dc:title xml:lang="en-US">RDF 1.1 XML Syntax</dc:title>
  </rdf:Description>

  <rdf:Description rdf:about="http://example.org/buecher/baum" xml:lang="de">
    <dc:title>Der Baum</dc:title>
    <dc:description>Das Buch ist außergewöhnlich</dc:description>
    <dc:title xml:lang="en">The Tree</dc:title>
  </rdf:Description>

</rdf:RDF>
```

4.1.5 JSON-LD

JSON-LD is a lightweight linked data format. It is easy for humans to read and write. It is based on the successful JSON format and provides a method to help interoperate with JSON data. JSON-LD is an ideal data format for programming environments, REST Web services, and unstructured databases such as Apache CouchDB and MongoDB. Linked data enables people to publish and use information on the web. It is a method of creating standards-based, readable data on websites. It allows applications to start with a piece of linked data and link to other linked data hosted on different sites on the web using embedded links. For a more detailed description of JSON-LD files, please refer to JSON-LD. Examples are as follows:

```
{
  "@context": {
    "name": "http://xmlns.com/foaf/0.1/name",
    "knows": "http://xmlns.com/foaf/0.1/knows"
  },
  "@id": "http://me.markus-lanthaler.com/",
  "name": "Markus Lanthaler",
  "knows": [
    {
      "name": "Dave Longley"
    }
  ]
}
```

4.2 Initialize the system database

As soon as you download and compile the code for the gStore system, a database named System (the real directory name system.db) is automatically created. This is the database that manages system statistics, including all users and all databases. You can query this database using the gquery command, but do not modify it using the editor.

The `system` database is the built-in system database of gStore. This database cannot be deleted. It is used to store the information related to the system, especially the information of the built database

4.2.1 Command line mode(ginit)

ginit is used to initialize the database

instruction:

```
bin/ginit -db [db_name1],[db_name2],[...]
```

command parameter:

```
db_name1: database name
```

If no database names are written, the reinitialized `system` database will have no other database information

example:

```
[root@localhost gStore]$ bin/ginit -db lubm
The database lubm is not exist, now create it.
...
system.db init successfully! Used 24 ms
```

4.3 Create database

Creating a database is one of the most important operations in gStore. It is also the first operation that users need to perform after gStore installation. gStore provides multiple methods to create a database.

4.3.1 Command line mode (gbuild)

The gbuild command is used to create a new database from an RDF file:

```
bin/gbuild -db dbname -f filename
```

Parameter definition:

```
dbname: database name
filename: filepath of files with ".nt" or ".n3" as suffix ,filepath of files
with ".zip "as suffix containing files with ".nt" or ".n3" as suffix
```

For example, we build a database called "lubm.db" from lubm.nt, which can be found in the data folder.

```
[root@localhost gStore]$ bin/gbuild -db lubm -f ./data/lubm/lubm.nt
database path: ./dbhome/lubm.db
...
Save the database info to system database....
...
Add database info success, update num : 3
Build RDF database lubm successfully! Used 2869 ms
```

Note:

- You cannot create a database from an empty RDF dataset
- Note that you cannot directly `CD` into the `bin` directory. Instead, you need to perform the gbuild operation in the gStore installation root director

4.3.2 Visual tool (gWorkbench)

gWorkbench is a visual management tool of gStore. Through gWorkbench, you can connect to gStore and create a graph database through database management module. For details, see [Development document] - [Visual Tool Workbench] - [query function] - [database management] function].

4.3.3 HTTP API (ghttp)

gStore provides GHTTP component as HTTP API service component. Users can realize relevant functions by sending HTTP requests to ghttp In ghttp, graph database is constructed by `build` request.

4.3.4 Socket API (gServer)

gStore provides gServer component as Socket API service component. Users can realize related functions by sending Socket request to gServer. gServer builds graph database through `build` request. See [Development document] - [Common API] - [gServer interface Description] for details.

4.4 Database list

The database list function is used to obtain information about all available databases in the following formats

4.4.1 Command line mode (gshow)

gshow is used to get the list of all available databases.

instruction:

```
bin/gshow
```

example:

```
[root@localhost gStore]$ bin/gshow
-------------------------------------------
| database | creator |     built_time      |
-------------------------------------------
| system   | root    |                     |
| small    | root    | 2024-08-08 10:24:01 |
| lubm     | root    | 2024-08-08 14:00:22 |
| num      | root    | 2024-08-08 10:31:42 |
| bbug     | root    | 2024-08-08 10:24:01 |
-------------------------------------------
```

4.4.2 Visual tool (gWorkbench)

gWorkbench is a visual management tool of gStore. Through gWorkbench, you can connect to gStore and create a graph database through database management module. For details, see [Development document] - [Visual Tool Workbench] - [query function] - [database management] function].

4.4.3 HTTP API (ghttp)

gStore provides ghttp component as http API service component, users can send http request to ghttp to achieve relevant functions, ghttp through the `show` command to achieve relevant functions, details see [development document] - [common API] - [ghttp interface description].

4.4.4 Socket API (gServer)

gStore provides gServer component as Socket API service component. Users can realize related functions by sending Socket request to gServer. gServer displays database list through `show` request. See [Development document] - [Common API] - [gServer interface Description] for details.

4.5 Database status Query

The database status query function is used to obtain statistics about a specified database in the following ways.

4.5.1 Command line mode (gmonitor)

gmonitor is used to get statistics about a specified database.

instruction:

```
bin/gmonitor -db db_name
```

parameter definition:

```
db_name: database name
```

example:

```
[root@localhost gStore]$ bin/gmonitor -db test1
----------------------------------------
|     name      |        value         |
----------------------------------------
| database      | lubm                 |
| creator       | root                 |
| built_time    | 2024-08-08 14:00:22 |
| triple_num    | 99550                |
| entity_num    | 28413                |
| literal_num   | 0                    |
| subject_num   | 16196                |
| predicate_num | 17                   |
| disk_used     | 21 MB                |
----------------------------------------
```

4.5.2 Visual tool（gWorkbench)

gWorkbench is a visual management tool of gStore. Through gWorkbench, you can connect to gStore and create a graph database through database management module. For details, see [Development document] - [Visual Tool Workbench] - [query function] - [database management] function].

4.5.3 HTTP API（ghttp)

gStore provides ghttp component as http API service component. Users can realize relevant functions by sending http requests to ghttp. In ghttp, database statistics can be obtained through `monitor`.

4.6 Database query

Database query is one of the most important functions of gStore. gStore supports SPARQL 1.1 query language defined by W3C. Users can use the gStore database query function in the following ways.

4.6.1 Command line mdoe（gquery)

gquery is used to query existing databases using files containing SPARQL queries. (Each file contains an exact SPARQL statement. SPARQL statements can be used not only for queries, but also for additions and deletions. For details on SPARQL statements, see Chapter 8.)

1.To query the database named db name, enter the following command:

```
bin/gquery -db db_name -q query_file
```

parameter definition：

```
db_name: database name
query_file：The path to the SPARQL statement ending with ".sql "(other name
extensions are acceptable)
```

For example, we execute the SPARQL statement in./data/lubm/ lubM_q0.sql to query the LUBM database.

The query result is:

```
[root@localhost gStore]$ bin/gquery -db lubm -q ./data/lubm/lubm_q0.sql
-------------------------------------------------------------
|                            ?x                             |
-------------------------------------------------------------
| <http://www.Department0.University0.edu/FullProfessor0>   |
| <http://www.Department1.University0.edu/FullProfessor0>   |
| <http://www.Department2.University0.edu/FullProfessor0>   |
| <http://www.Department3.University0.edu/FullProfessor0>   |
| <http://www.Department4.University0.edu/FullProfessor0>   |
| <http://www.Department5.University0.edu/FullProfessor0>   |
| <http://www.Department6.University0.edu/FullProfessor0>   |
| <http://www.Department7.University0.edu/FullProfessor0>   |
| <http://www.Department8.University0.edu/FullProfessor0>   |
| <http://www.Department9.University0.edu/FullProfessor0>   |
| <http://www.Department10.University0.edu/FullProfessor0>  |
| <http://www.Department11.University0.edu/FullProfessor0>  |
| <http://www.Department12.University0.edu/FullProfessor0>  |
| <http://www.Department13.University0.edu/FullProfessor0>  |
| <http://www.Department14.University0.edu/FullProfessor0>  |
-------------------------------------------------------------
query database successfully, Used 72 ms
```

2.To learn more about how to use gQuery, enter the following command:

```
bin/gquery --help
```

3.Command to enter gquery console:

```
bin/gquery -db dbname
```

he program displays a command prompt (" gsql > ") where you can enter command:

> Use `help` to see the basic information for all commands

> Enter `quit` to exit the gquery console

> For the `sparql` command, use `sparQL query_file` to execute the SPARQL query statement, and query_file indicates the path of the file storing the SPARQL statement. When the program finishes answering the query, it displays the command prompt again
>
> We use lubm.nt as example。

```
(base) [root@iz8vb0u9hafhzz1mn5xcklz gStore]# bin/gquery -db lubm

gsql>sparql ./data/lubm/lubm_q0.sql
query is:
select ?x where
{
        ?x      <ub:name>       <FullProfessor0>.
}
```

```
-------------------------------------------------------------
|                            ?x                             |
-------------------------------------------------------------
| <http://www.Department0.University0.edu/FullProfessor0>  |
| <http://www.Department1.University0.edu/FullProfessor0>  |
| <http://www.Department2.University0.edu/FullProfessor0>  |
| <http://www.Department3.University0.edu/FullProfessor0>  |
| <http://www.Department4.University0.edu/FullProfessor0>  |
| <http://www.Department5.University0.edu/FullProfessor0>  |
| <http://www.Department6.University0.edu/FullProfessor0>  |
| <http://www.Department7.University0.edu/FullProfessor0>  |
| <http://www.Department8.University0.edu/FullProfessor0>  |
| <http://www.Department9.University0.edu/FullProfessor0>  |
| <http://www.Department10.University0.edu/FullProfessor0> |
| <http://www.Department11.University0.edu/FullProfessor0> |
| <http://www.Department12.University0.edu/FullProfessor0> |
| <http://www.Department13.University0.edu/FullProfessor0> |
| <http://www.Department14.University0.edu/FullProfessor0> |
-------------------------------------------------------------

gsql>help
help - print commands message
quit - quit the console normally
sparql - load query from the second argument

gsql>quit
```

- Note:

  - If there is no answer, "[Empty Result]" is printed with a blank line following all results.

  - Use readline lib, so you can use the arrow keys in the keyboard to view the command history, and use the and arrow keys to move and modify the entire command.

  - Utility supports path completion. (Not built-in command completion)

  - Note that you cannot directly 'CD' into the 'bin' directory, but rather perform the 'gquery' operation in the gStore installation root directory

4.6.2 Visual tool (gWorkbench)

gWorkbench is a visual management tool of gStore. Through gWorkbench, you can connect to gStore and create a graph database through database management module. For details, see [Development document] - [Visual Tool Workbench] - [query function] - [database management] function].

4.6.3 HTTP API（ghttp)

gStore provides ghttp component as http API service component, users can realize related functions by sending http request to ghttp, ghttp through `query` request to query graph database, including query, delete, insert, See [Development document] - [common API] - [ghttp interface Description] for details.

4.6.4 Socket API（gServer)

gStore provides gServer component as Socket API service component. Users can realize related functions by sending Socket request to gServer. gServer can query graph database through `query` request, including query, delete and insert. See [Development document] - [Common API] - [gServer interface Description] for details.

4.7 Database export

The export database function enables you to export a database as an. Nt file. There are three forms:

4.7.1 Command line mode (gexport)

gexport is used to export database.

instruction:

```
bin/gexport -db db_name -f path
```

parameter definition:

```
db_name: database name
path: Export to specified folder (if empty, export to gStore root by default)
```

example:

```
[root@localhost gStore]# bin/gexport -db lubm
gexport...
...
start exporting the database......
...
finish loading
lubm.db exported successfully! Used 187 ms
lubm.db export path: lubm_240808141125.nt
```

4.7.2 Visual tool (gWorkbench)

gWorkbench is a visual management tool of gStore. Through gWorkbench, you can connect to gStore and create a graph database through database management module. For details, see [Development document] - [Visual Tool Workbench] - [query function] - [database management] function].

4.7.3 HTTP API (ghttp)

gStore provides ghttp component as httpAPI service component. Users can realize relevant functions by sending http requests to ghttp. The function of `export` is adopted in ghttp.

4.8 Database deletion

Deleting a database You can delete a database in the following three ways

4.8.1 Command line mode (gdrop)

gdrop is use to delete database

instruction:

```
bin/gdrop -db db_name
```

parameter definition:

```
db_name：database name
```

example:

```
[root@localhost gStore]$ bin/drop -db lubm2
Begin to drop database....
...
Database lubm2 dropped successfully! Used 20 ms
```

To delete the database, you should not just type `rm -r db_name. Db` because this will not update the built-in database named `system`. Instead, you should type `bin/ gdrop-db db_name`.

4.8.2 Visual tool（gWorkbench)

gWorkbench is a visual management tool of gStore. Through gWorkbench, you can connect to gStore and create a graph database through database management module. For details, see [Development document] - [Visual Tool Workbench] - [query function] - [database management] function].

4.8.3 HTTP API（ghttp)

gStore provides ghttp component as http API service component, users can send http request to ghttp to achieve relevant functions, ghttp through the `drop` command to achieve relevant functions, details see [development document] - [common API] - [ghttp interface description]

4.8.4 Socket API（gServer)

gStore provides gServer component as Socket API service component. Users can realize related functions by sending Socket request to gServer. gServer can delete graph database by `drop` request. See [Development document] - [Common API] - [gServer interface Description] for details.

4.9 Additional data

Inserting RDF data is a routine gStore operation, and you can do it in one of the following ways.

4.9.1 Command line mode (gadd)--file

gadd is use to insert triples from a file into an existing database.

instruction:

```
bin/gadd -db db_name -f rdf_triple_file_path
```

parameter definition:

```
dbname: database name
filename: filepath of files with ".nt" or ".n3" as suffix ,filepath of files
with ".zip "as suffix containing files with ".nt" or ".n3" as suffix
```

example:

```
[bookug@localhost gStore]$ bin/gadd -db lubm -f ./data/lubm/lubm.nt
...
finish loading
after inserted triples num 0,failed num 0,used 2452 ms
```

Note:

1. Gadd is primarily used for data insertion into RDF files

2. Do not go directly to the 'bin' directory. Instead, perform the 'gadd' operation in the gStore installation root directory

4.9.2 Command line mode (gquery)--SPARQL statement

SPARQL can be defined by `insert data` instruction to insert data. Based on this principle, users can also write SPARQL insert statements and then use gStore's `gQuery` tool to insert data. Examples of SPARQL insert statements are as follows:

```
insert data {
    <Johnny> <gender> "male"^^<http://www.w3.org/2001/XMLSchema#String>.
    <Johnny> <age> "28"^^<http://www.w3.org/2001/XMLSchema#Int>.
    <Johnny> <friend> <Peter>.
}
```

Multiple RDF data can be contained with `{}`, taking care that each one ends with a `.`

Since the database query function can be used to insert data, the following functions can also be used to insert data.

4.9.3 Visual tool (gWorkbench)

gWorkbench is a visual management tool of gStore. Through gWorkbench, you can connect to gStore and create a graph database through database management module. For details, see [Development document] - [Visual Tool Workbench] - [query function] - [database management] function].

4.9.4 HTTP API (ghttp)

gStore provides ghttp component as AN http API service component. Users can realize relevant functions by sending http requests to ghttp. ghttp inserts data through `query` request and batch inserts data through `batchInsert`. See [Development document] - [common API] - [ghttp interface Description] for details.

4.9.5 Socket API（gServer）

gStore provides gServer component as Socket API service component. Users can implement related functions by sending Socket requests to gServer. gServer inserts data through 'query' requests. See [Development document] - [Common API] - [gServer interface Description] for details.

4.10 Data deletion

Deleting RDF data is a routine gStore operation, and you can perform this operation in the following ways.

4.10.1 Command line mode (gsub)--file

gsub is used to remove triples from files in an existing database.

instruction：

```
bin/gsub -db db_name -f rdf_triple_file_path
```

parameter definition：

```
rdf_triple_file_path: filepath of files with ".nt" or ".n3" as suffix ,filepath
of files with ".zip "as suffix containing files with ".nt" or ".n3" as suffix
```

example：

```
[root@localhost gStore]$ bin/gsub -db lubm -f ./data/lubm/lubm.nt
...
finish loading
after remove, used 2580 ms
```

4.10.2 Command line mode（gquery)---SPARQL statement

SPARQL can be defined by 'delete data' instruction to insert data. Based on this principle, users can also write SPARQL insert statements and then use gStore's `gQuery` tool to insert data. Examples of SPARQL insert statements are as follows:

```
delete data {
 <San Zhang> <gender> "male"^^<http://www.w3.org/2001/XMLSchema#String>.
<San Zhang> <age> "28"^^<http://www.w3.org/2001/XMLSchema#Int>.
<San Zhang> <friend> <Si Li>.
}
```

Multiple RDF data can be contained with '{}', taking care that each one ends with a `.`

SPARQL also allows you to delete data based on a subquery structure using the 'DELETE WHERE' statement, as shown below

```
delete where
{
   <San Zhang> ?x ?y.
}
```

This statement deletes all information (including attributes and relationships) about an entity

Since the database query function can be used to insert data, the following functions can also be used to insert data

4.10.3 Visual tool（gWorkbench)

gWorkbench is a visual management tool for gStore. Through gWorkbench, you can connect to gStore and delete data by writing SPARQL statements through database query module. See [Development document] - [Visualization Tool Workbench] - [query function] - [Graph database query] for details.

4.10.4 HTTP API（ghttp)

gStore provides ghttp component as http API service component, users can send http request toghttp to achieve relevant functions, ghttp through `query` request to delete data and `batchRemove` to delete data in batches. See [Development document] - [common API] - [ghttp interface Description] for details

4.10.5 Socket API（gServer)

gStore provides gServer component as Socket API service component. Users can realize related functions by sending Socket requests to gServer. gServer can delete data through `query` requests. See [Development document] - [Common API] - [gServer interface Description] for details.

4.11 Console Service

gStore provides a console service with contextual information via the gconsole module, which users can use to perform all types of database management operations.

4.11.1Start gconsole

(1) Login

```
bin/gconsole [-u <usr_name>]
```

A prompt for input will appear if `usr_name` is missing.

```
# bin/gconsole
Enter user name: root
```

```
Enter password:

Gstore Console , an interactive shell based utility to communicate with gstore
repositories.
Gstore version: 1.3 Source distribution
Copyright (c) 2016, 2024, pkumod and topgraph and/or its affiliates.

Welcome to the Gstore Console.
Commands end with ;. Cross line input is allowed.
Comment start with #. Redirect (> and >>) is supported.
CTRL+C to quit current command. CTRL+D to exit this console.
Type 'help;' for help.

gstore>
```

(2) Help

```
bin/gconsole --help
```

```
# bin/gconsole --help
Gstore Ver 1.3 for Linux on x86_64 (Source distribution)
Gstore Console(gconsole), an interactive shell based utility to communicate with
gstore repositories.
Copyright (c) 2016, 2024, pkumod and topgraph and/or its affiliates.

Usage: bin/gconsole [OPTIONS]
  -?, --help          Display this help and exit.
  -u, --user          username.

Supported command in gconsole: Type "?" or "help" in the console to see info of
all commands.
```

(3)Commands

- Ends with `;` and can span multiple lines
- Supports using `>` and `>>` to redirect output
- Supports single-line comments starting with `#` until a new line
- Supports command completion, file name completion, line editing and history commands viewing
- Abandon the current command (end execution or abandon editing) with `ctrl+C`, exit the command line with `ctrl+D`
- To enter gconsole, you need to log in; only commands that meet the current user permissions will be executed, and only the content that meets the current user permissions will be displayed.
  - 7 types of permissions: `query`, `load`, `unload`, `update`, `backup`, `restore`, `export`
- Command keywords are case-insensitive. Command-line options, database names, user names, and passwords are case-sensitive. Command keywords as follows:
  - Database management: `sparql`, `create`, `use`, `drop`, `show`, `showdbs`, `backup`, `restore`, `export`, `pdb`
  - Identity: `flushpriv`, `pusr`, `setpswd`
  - Configuration information viewing: `settings`, `version`

- Permissions management: `setpriv`, `showusrs`, `pusr`
- User management: `addusr`, `delusr`, `setpswd`, `showusrs`
- Help and miscellaneous: `quit`, `help/?`, `pwd`, `clear`

4.11.2 Database management

(1)Build database

```
create <database_name> [<nt_file_path>]
```

- Read `.nt` file from nt_file_path and create a database named database_name, or create an empty database
- <database_name>: cannot be `system`
- <nt_file_path>: file path

```
gstore> create example friend.nt;
... (this is build database process output, omitted in this document)
Add database info success.
Build RDF database example successfully!
```

(2)Delete database

```
drop <database_name>
```

- Cannot delete the current database
- <database_name>: can only be specified as the name of a database for which the current user has full permissions

```
gstore> drop <database_name>;
```

(3)Load/Switch database

```
use <database_name>
```

- Load/switch the current database: unload the previous current database from memory (if any) and load the specified database
- The target database needs to have been created
- <database_name>: can only be specified as the name of a database for which the current user have load and unload permissions

```
gstore> use example;
... (this is load process output, omitted in this document)
Current database switch to example successfully.
```

(4)Query/update database

- Run a SPARQL query on the current database
- You need to specify the current database first by `use <database_name>`

(5)Inputting a SPARQL query

```
gstore> # show all in db
```

```
     -> SELECT ?x ?y ?z
     -> WHERE{
     ->     ?x ?y ?z. # comment
     -> } ;
... (this is query process output, omitted in this document)
---------------------------------------
|    ?x     |     ?y      |    ?z     |
---------------------------------------
| <Alice>   | <关注>      | <Bob>     |
| <Bob>     | <关注>      | <Alice>   |
| <Carol>   | <关注>      | <Bob>     |
| <Dave>    | <关注>      | <Alice>   |
| <Dave>    | <关注>      | <Eve>     |
| <Alice>   | <喜欢>      | <Bob>     |
| <Bob>     | <喜欢>      | <Eve>     |
| <Eve>     | <喜欢>      | <Carol>   |
| <Carol>   | <喜欢>      | <Bob>     |
| <Francis> | <喜欢>      | <Carol>   |
| <Alice>   | <不喜欢>    | <Eve>     |
| <Carol>   | <不喜欢>    | <Francis> |
| <Francis> | <不喜欢>    | <Eve>     |
| <Francis> | <不喜欢>    | <Dave>    |
| <Dave>    | <不喜欢>    | <Francis> |
---------------------------------------
query database successfully, Used 127 ms
```

Example of redirection (all commands support redirecting output):

```
gstore> # support redirect
    -> SELECT ?x ?y ?z
    -> WHERE{
    -> ?x ?y ?z.
    -> } > my_test_data/output ;
Redirect output to file: my_test_data/output
```

(6)sparql <sparql_file>

```
sparql sparql_file
## sparql_fileis a SPARQL query file, which can be either a relative path or an
absolute path.
```

- Specify the sparql file: the file contains multiple sparql statements that need to be separated by `;` and the last `;` is optional
- The file content supports single-line comments starting with #

```
gstore> sparql all.sparql ;
... (this is query process output, omitted in this document)
-----------------------------------
|    ?s     |    ?p    |   ?o     |
-----------------------------------
| <Alice>   | <关注>   | <Bob>    |
| <Bob>     | <关注>   | <Alice>  |
| <Carol>   | <关注>   | <Bob>    |
```

```
| <Dave>    | <关注>   | <Alice> |
| <Dave>    | <关注>   | <Eve>   |
| <Alice>   | <喜欢>   | <Bob>   |
| <Bob>     | <喜欢>   | <Eve>   |
| <Eve>     | <喜欢>   | <Carol> |
| <Carol>   | <喜欢>   | <Bob>   |
| <Francis> | <喜欢>   | <Carol> |
----------------------------------
query database successfully, Used 121 ms
```

- An example of query.sparql content:

```
SELECT  ?x ?y ?z
WHERE {
    ?x ?y ?z.
}
```

(7)Display database information

```
show [<database_name>] [-n <displayed_triple_num>]
```

- Display meta information and the first displayed_triple_num triples (the first 10 triples are displayed by default)
- <database_name>: the database name. If database_name is not specified, the information of the current database will be displayed.
- -n <displayed_triple_num>: number of lines displayed

```
gstore> show example;
... (this is load and query process output, omitted in this document)
---------------------------------------
|     name      |        value        |
---------------------------------------
| database      | example             |
| creator       | root                |
| built_time    | 2024-08-06 17:23:47 |
| triple_num    | 15                  |
| entity_num    | 6                   |
| literal_num   | 0                   |
| subject_num   | 6                   |
| predicate_num | 3                   |
---------------------------------------
----------------------------------
|    ?s     |    ?p    |   ?o    |
----------------------------------
| <Alice>   | <关注>   | <Bob>   |
| <Bob>     | <关注>   | <Alice> |
| <Carol>   | <关注>   | <Bob>   |
| <Dave>    | <关注>   | <Alice> |
| <Dave>    | <关注>   | <Eve>   |
| <Alice>   | <喜欢>   | <Bob>   |
| <Bob>     | <喜欢>   | <Eve>   |
| <Eve>     | <喜欢>   | <Carol> |
| <Carol>   | <喜欢>   | <Bob>   |
| <Francis> | <喜欢>   | <Carol> |
----------------------------------
```

(8)View all databases

```
showdbs
```

- Only the databases for which the current user has query permissions can be displayed.

```
gstore> showdbs;
---------------------------------------
| database | creater |    status     |
---------------------------------------
| system   | root    |               |
| test1    | root    | already_built |
| test     | root    | already_built |
| test2    | root    | already_built |
| test3    | root    | already_built |
| example  | root    | already_built |
```

(9)Display current database name

```
pdb
```

```
gstore> pdb;
example
```

(10)Backup current database

```
backup [<backup_folder>]
```

- <backup_folder> specifies the backup path.

```
gstore> backup;
... (this is backup process output, omitted in this document)
Database example backup successfully.
```

```
gstore> backup back;
... (this is backup process output, omitted in this document)
Database example backup successfully.
```

(11)Export current database

```
export <file_path>
```

- <file_path> specifies the export path.

```
gstore> export example.nt;
Database example export successfully.
```

(12)Restore database

```
restore <database_name> <backup_path>
```

- <database_name>: database name.
- <backup_path>: backup file path.

```
gstore> restore example backups/example.db_220929114732/
... (this is restore process output, omitted in this document)
Database example restore successfully.
```

4.11.3 User Identity

(1)Refresh permissions

```
flushpriv
```

Read db and refresh current user permissions

```
gstore> flushpriv;
Privilige Flushed for current user successfully.
```

(2)Display current username and permissions

```
pusr [<database_name>]
```

- `pusr` displays the current username
- `pusr <database_name>` displays the current user name and the current user's permissions on <database_name>

```
gstore> pusr;
usrname: yuanzhiqiu

gstore> pusr example;
usrname: yuanzhiqiu
privilege on eg: query load unload update backup restore export
```

(3)Modify the current user's password

```
setpswd
```

- Password verification is required.

```
gstore> setpswd;
Enter old password:
Enter new password:
Enter new password again:
Not Matched.
Enter new password:
Enter new password again:
Password set successfully.
```

4.11.4 Configuration information viewing

(1)View congifuration

```
settings [<conf_name>]
```

- Show all configuration information.

```
gstore> settings;
Settings:
thread_num      30
... (this is other settings, omitted in this document)
You can Edit configuration file to change settings: ./conf/conf.ini
```

- <conf_name> displays the configuration information of the specified conf_name.

```
gstore> settings ip_deny_path;
"./conf/ipDeny.config"
You can Edit configuration file to change settings: ./conf/conf.ini
```

(2)View version

```
version
```

- Output current version information.

```
gstore> version;
Gstore version: 1.3 Source distribution
Copyright (c) 2016, 2024, pkumod and topgraph and/or its affiliates.
```

4.11.5Permissions management

Only system users have the permission to execute commands related to permissions management. The permissions descriptions are as follows:

- Permissions: the operations a user is permitted to carry out on a database
- 7 types of permissions: `query`, `load`, `unload`, `update`, `backup`, `restore`, `export`

  ```
  [1]query [2]load [3]unload [4]update [5]backup [6]restore [7]export
  ```

Users who have all seven permissions are considered to have full permissions.

- System users have full permissions on all databases. (System users are defined in conf.ini and system.db.)

(1)Set user's permissions

```
setpriv <usrname> <database_name>
```

- A password is required to verify root identity.
- <usrname>: the user name to be set.
- <database_name>: database name.

```
gstore> setpriv zero eg;
Enter your password:
[1]query [2]load [3]unload [4]update [5]backup [6]restore [7]export [8]all
Enter privilege number to assign separated by whitespace:
1 2 3
[will set priv:]00001110
Privilege set successfully.
```

(2)View user's permissions

```
pusr <database_name> <usrname>
```

- <usrname> : the user name.
- <database_name>: database name.

```
gstore> pusr eg yuanzhiqiu;
privilege on eg: query load unload update backup restore export
```

(3)View all users and their permissions

```
showusrs
```

Show which users there are and the database permissions for each user (non-root users only; only databases with has_xxx_priv for this user are displayed.)

Output format:

```
usr
------------------------
privilege on databases
```

```
-------------------------------------------------------------
| user |                    privilege                      |
-------------------------------------------------------------
| root | all privilege on all db                           |
| uki  | unload: query load unload update backup export    |
-------------------------------------------------------------
```

4.11.6 User management

Only system users have the permission to execute user management commands.

(1)Add users

```
addusr <usrname>
```

- A password is required to verify root identity.
- <usrname>: the new user name.

```
gstore> addusr uki;
Enter your password:
Enter password for new user: hello
Add usr uki successfully.
```

(2)Delete users

```
delusr <usrname>
```

- A password is required to verify root identity.
- <usrname>: the user name to be deleted.

```
gstore> delusr cat;
Enter your password:
Del usr cat successfully.
```

(3)Reset passwords

```
setpswd <usrname>
```

- A password is required to verify root identity.
- <usrname>: user name.

```
gstore> setpswd zero;
Enter your password:
Enter new password:
Enter new password again:
Password set successfully.
```

(4)View all users

```
showusrs
```

```
-------------------------------------------------------------
| user |                    privilege                       |
-------------------------------------------------------------
| root | all privilege on all db                            |
| uki  | unload: query load unload update backup export     |
-------------------------------------------------------------
```

4.11.7 Help and miscellaneous

(1)End/Cancel the current command

`ctrl+C`

End the command that is currently being executed:

```
...(executing some cmd) (ctrl+C here)
gstore>
```

Or discard the command you are currently typing:

```
gstore> SELECT ?x ?y ?z
-> WHERE{ (ctrl+C here)
gstore>
```

(2)Quit

`quit` / `ctrl+D`

This will unmount the current database from memory if it is not yet unmounted.

```
gstore> quit;
[xxx@xxx xxx]#
```

```
gstore> (ctrl+D here)
[xxx@xxx xxx]#
```

(3)Help

```
help [edit/usage/<command>]
```

- `help` displays the help information about all commands:

```
gstore> help;
Gstore Console(gconsole), an interactive shell based utility to communicate with
gStore repositories.

For information about gStore products and services, visit:
   http://www.gstore.cn/
For developer information, including the gStore Reference Manual, visit:
   http://www.gstore.cn/pcsite/index.html#/documentation

Commands end with ;. Cross line input is allowed.
Comment start with #.
Type 'cancel;' to quit the current input statement.

List of all gconsole commands:
sparql          Answer SPARQL query(s) in file.
create          Build a database from a dataset or create
...(this is help msg for other commands, omitted in this document)

Other help arg:
edit    Display line editing shortcut keys supported by gconsole.
usage   Display all commands as well as their usage.
```

`help edit` displays the line editing shortcut operations of `GNU readline`:

```
gstore> help edit;
```

```
Frequently used GNU Readline shortcuts:
CTRL-a  move cursor to the beginning of line
CTRL-e  move cursor to the end of line
CTRL-d  delete a character
CTRL-f  move cursor forward (right arrow)
CTRL-b  move cursor backward (left arrow)
CTRL-p  previous line, previous command in history (up arrow)
CTRL-n  next line, next command in history (down arrow)
CTRL-k  kill the line after the cursor, add to clipboard
CTRL-u  kill the line before the cursor, add to clipboard
CTRL-y  paste from the clipboard
ALT-b   move cursor back one word
ALT-f   move cursor forward one word
For more about GNU Readline shortcuts, see
https://en.wikipedia.org/wiki/GNU_Readline#Emacs_keyboard_shortcuts
```

`help usage` displays the description and usage of all commands:

```
gstore> help usage;
List of all gconsole commands:
Note that all text commands must be end with ';' but need not be in one line.
sparql          Answer SPARQL query(s) in file.
                sparql <; separated SPARQL file>;
create          Build a database from a dataset or create an empty database.
                create <database_name> [<nt_file_path>];
...(this is help msg for other commands, omitted in this document)
```

`help <command>` displays the help information about the command:

```
gstore> help use;
use     Set current database.
        use <database_name>;
```

(4)Clear screen

```
gstore> clear;
```

(5)View current working path

```
gstore> pwd;
/<path_to_gstore>/gstore
```

4.12 HTTP API service

GStore provides two sets of http API services, namely grpc and ghttp. Users can remotely connect to and operate on gStore by sending http requests.

4.12.1 Starting ghttp/gRPC service

After gStore is compiled, a ghttp service and a gRPC service is displayed in the bin directory of gStore. However, the service is not started by default. You need to manually start the service:

```
nohup bin/grpc -db db_name -p serverPort -c enable &
```

```
nohup bin/ghttp -db db_name -p serverPort -c enable &
```

Parameters:

```
db_name: The name of the database to start HTTP (Optional; The effect of this
parameter is that when starting the HTTP service, the related information of the
database will be loaded into memory.)

serverPort: the HTTP listening port. You need to ensure that this port is not
prohibited by the server firewall. (Optional; if this port is not specified, the
default port is 9000).

enable: whether to load CSR resources. 0 indicates no and 1 indicates yes.
(Optional, for built-in advanced functions or user-defined graph analysis
functions. The default value is 0.)
```

`ghttp` and `grpc` both support GET and POST request types.

Concurrent read-only queries are supported, but when a query containing updates arrives, the entire database is locked. On a computer with dozens of kernel threads, the recommended number of queries to run concurrently is less than 300, but we were able to run 13,000 queries simultaneously in our experiment. To use the concurrency feature, it is best to set the number of open files and the maximum number of processes to 65535 or greater in the system settings.

It is recommended to periodically execute the `checkpoint` command to the console if you frequently send query containing updates. Otherwise, updates may not be synced with the disk, and will be lost if the server stops abnormally (for example, because of typing "Ctrl + C")

4.12.2 Shutting down the service

For safety, please use the following command to shut down the service instead of `Ctrl + C` or `kill`.

```
bin/shutdown
```

4.12.3 HTTP API

ghttp and gRPC provide a rich API interface so that users can remotely operate most of the functions of gStore. For details, please see [Development document] - [common API] - [ghttp interface Description] or [gRPC interface Description].

4.13 Socket API service

gServer is an external access interface provided by gStore and a Socket API service. Users can connect and operate gStore remotely through two-way socket communication.

4.13.1 Start gServer service

After gStore is compiled, a gServer service is displayed in the bin directory of gStore. However, the gServer service is not started by default. You need to manually start the gServer service:

```
bin/gserver -s
```

Other optional parameters:

```
-t,--stop: shutdown gserver service;
-r,--restart：restart gserver service；
-p,--port：Modify the socket connection port configuration. The default port is
9000. After the modification, restart the GServer service
-P,--printport：pritn current socket connection port configuration
-d,--debug：start debug mode（keep gserver service running in the foreground）
-k,--kill：Forcibly stop the service. You are advised to stop the service only
when the service cannot be stopped
```

4.13.2 Shutdown gServer service

To stop the gServer service, use the following command to stop it. It is better not to simply enter the command 'Ctrl + C' or 'kill' to stop the gServer, because it is not safe.

```
bin/gserver -t
```

4.13.3 gServer related API

gServer provides rich API interfaces so that users can remotely operate most functions of gStore. See [Development document] - [common API] - [gServer interface Description] for specific interfaces.

5. API Usage

5.1 Introduction to API

gStore provides API services to users through http and Socket services, and its components are gRPC, ghttp and gServer.

5.1.1 Introduction to HTTP API

We now provide C++, Java, Python, PHP, and Node.js APIs for gRPC and ghttp. Please refer to the examples in `api/http/cpp`, `api/http/java`, `api/http/python`, `api/http/php` and `api/http/nodejs`. To use these examples, make sure you have generated the executable files of gRPC or ghttp. **Next, please use the `bin/grpc` or `bin/ghttp` command to start the service.** If you know of a running gRPC or ghttp server that's available for connection, you can also directly connect to it. Then, for the C++ and Java code, you need to compile the sample code in the directory `api/http/cpp/example` or `api/http/java/example`.

HTTP API Framework

The gStore HTTP API is placed in the API/HTTP directory of the gStore root directory and contains the following:

- gStore/api/http/
  - cpp/ (the C++ API)
    - example/ (Example program using the C++ API)
      - Benchmark.cpp
      - GET-example.cpp
      - POST-example.cpp
      - Transaction-example.cpp
      - Makefile (for compiling the sample code)
    - src/
      - GstoreConnector.cpp (C++ API's source code)
      - GstoreConnector.h
      - Makefile (for compiling the lib)
  - java/ (the Java API)
    - example/ (Example program using the Java API)
      - Benckmark.java
      - GETexample.java
      - POSTexample.java
      - Makefile
    - src/
      - jgsc/
        - GstoreConnector.java (Java API's source code)
      - Makefile
  - python/ (the Python API)
    - example/ (Example program using the Python API)
      - Benchmark.py
      - GET-example.py

        - POST-example.py
      - src/
        - GstoreConnector.py
    - nodejs/ (the Node.js API)
      - example/ (Example program using the Node.js API)
        - POST-example.js
        - GET-example.js
      - src
        - GstoreConnector.js (Node.js API's source code)
        - LICENSE
        - package.json
    - php/ (the PHP API)
      - example/ (Example program using the PHP API)
        - Benchmark.php
        - POST-example.php
        - GET-example.php
      - src/
        - GstoreConnector.php (the PHP API's source code)
- -

For details on how to start and shut down gRPC or ghttp, please see [development documentation] - [Quick Start] - [HTTP API service].

After the API service is started, the gRPC access address is as follows:

```
http://serverip:port/grpc/
```

The ghttp access address is as follows:

```
http://serverip:port/
```

`ServerIP` is the IP address of the gStore server, and `port` is the port on which gRPC or ghttp is started.

5.1.2 Introduction to Socket API

We now provide C++ (Java, Python, PHP, and Node.js will be supported in future versions) API for gServer, the socket API service. Please see api/socket/cpp for the example code. To use these examples, make sure you have generated an executable file for gServer. **Next, use the `bin/gserver -s` command to start the gserver service.** If you know of a running gServer that's available for connection, you can also directly connect to it. Then, for the C++ code, you need to compile the sample code in the directory `api/socket/cpp/example`.

For details on how to start and shut down gServer, please see [development documentation] - [Quick Start] - [Socket API service].

After the Socket API is started, you can connect through the Socket. The default port of gServer is 9000.

Socket API Framework

gStore的Socket API放在gStore根目录的API/Socket目录中，其内容如下:
The gStore Socket API is placed in the API/Socket directory of the gStore root directory and contains the following:

- gStore/api/socket/
  - cpp/ (the C++ API)
    - example/ (Example program using the C++ API)
      - CppAPIExample.cpp
      - Makefile ((for compiling the sample code)
    - src/
      - Client.cpp (C++ API's source code)
      - Client.h
      - Makefile (for compiling the lib)

5.2 http API Instruction

5.2.1 ghttp Service Interconnection Mode

> The ghttp interface adopts the `HTTP` protocol and supports multiple ways to access the interface. If the ghttp is started on the port `9000`, the interface interconnection content is as follows
> API address:
>
> ```
> http://ip:9000/
> ```
>
> The interface supports both `GET` and `POST` requests, where `GET` requests place > > parameters in the URL and `POST` requests place parameters in the `body` request.
>
> **Note: `GET` request parameters contain special characters, such as? , @,& and other characters, you need to use urlencode encoding, especially the `SPARQL` parameter must be encoded**

5.2.2 grpc Service Interconnection Mode

> The grpc interface adopts the HTTP protocol and supports multiple ways to access the interface. If the grpc is started on the port 9000, the interface connection information is as follows.
>
> API address:
>
> ```
> http://ip:9000/grpc
> ```
>
> The interface supports both `GET` and `POST` requests, where `GET` requests place parameters in the URL and `POST` requests place parameters in the `body` request or use `form` to express a request.
>
> Post request method 1 (recommended) : the parameter is passed as `JSON` structure by `raw` in `body` in `httprequest` (Requires RequestHeader's parameter Content-Type to be application/json)

> Post request method 2: the parameter is passed as a form (Requires RequestHeader's parameter Content-Type to be application/x-www-form-urlencoded)
>
> **Note: If the `GET` request parameters contain special characters, such as? , @, and &, you need to use the urlencode encoding, especially for the `SPARQL` parameter.**

5.2.3 API List

> The red font represents the latest version additions, while the blue font represents the latest version modifications

API name	Definition	Note
Interface of system		
check	heartbeat signal	Detect ghttp heartbeat signal
login	login to database	It is used to authenticate user names and passwords

API name	Definition	Note
testConnect	testing connectivity	Used to check whether GHTTP is connected
getCoreVersion	get gStore version	Get the gStore version number
ipmanage	black/white list management	Maintains a blacklist and whitelist of IP addresses that access gStore
upload	upload files	support file types to upload are nt, ttl, n3, rdf, txt.
download	download files	support to download files in gstore main directory and its sub directories.
stat	query system resources	statistics CPU, memory, disk available space information.
shutdown	close ghttp	
Interface of system operations		
show	display graph database	Display a list of all databases
load	load graph database	Load the database into memory
unload	unload graph database	Unload the database from memory
monitor	monitor graph database	Count information about the specified database (such as the number of triples, etc.)
build(update)	build graph database	The database file must be locally stored on the server
drop	drop graph database	Logical deletion and physical deletion can be performed
backup(update)	backup database	backup database information
backuppath	get the backup database path	Returns a list of all backup files in the default backup path.
restore(update)	restore database	restore database information
query	query database	Including query, delete, and insert
export	export database	Export database as NT file

API name	Definition	Note
batchInsert(update)	batch insert data	Batch insert NT data
batchRemove(update)	batch delete data	Batch delete NT data
rename	rename graph database	modify graph database name.
checkOperationState(new)	Query asynchronous operation status	Query the asynchronous execution status of build, backup, restore, batch_insert, batch_remove
Interface of database transaction		
begin	Start transaction	Transaction starts and needs to be used in conjunction with TQuery
tquery	Querying the database (with transactions)	Data queries with transaction mode (insert and DELETE only)
commit	commit transactions	Commit the transaction after it completes
rollback	rollback transaction	Roll back the transaction to begin state
checkpoint	write data to a disk	After an INSERT or delete operation is performed on the database, manually checkpoint is required
Interface of user management		
showuser	display all user list	Display a list of all users
usermanage	user management	Add, delete, or modify user information
userprivilegemanage	user privilege management	Add, delete, or modify user's privilege information
userpassword	modify user's password	modify user's password
Interface of custom function		
funquery	Query operator function	Gets a list of user-defined graph analysis functions

API name	Definition	Note
funcudb	Manage operator function	Adds, modifies, deletes, or compiles operator function
funreview	Preview operator function	View the latest generated graph analysis function source code
Interface of log		
txnlog	Obtain transaction log information	Returns transcation log information as JSON
querylogdate	Gets the date list of query logs	Returns the date list of existing query logs
querylog	Get query log information	Returns query log information as JSON
accesslogdate	Get the date of the API log	Returns the date list of existing API logs
accesslog	Get API access logs	Returns API access log information as JSON
Interface of reason engine(new)		
addReason	add reason	
listReason	list reason	
compileReason	compile reason	
executeReason	execute reason	
disableReason	disable reason	
showReason	show reason	
removeReason	remove reason	

5.2.4 API Specific Instruction

> The input and output parameters of each interface are specified in this section. Assume that the IP address of the GHTTP server is 127.0.0.1 and the port is 9000
> Interfaces of system

5.2.5 Interfaces of System

5.2.5.1 check

Brief description

- Detect ghttp heartbeat signal

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **check**

Return value

Parameter name	Tpye	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "the ghttp server is running..."
}
```

5.2.5.2 login

Brief description

- User login (verify username and password)
- Updates in this version: the full path information of RootPath will be returned after successful login

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `httprequest` , passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **login**
username	yes	string	User name

Parameter name	Mandatory	Type	Note
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
CoreVersion	string	Kernel version
licensetype	string	Certificate type (Open Source or Enterprise)
RootPath	string	Full path to the gStore root directory
type	string	HTTP service type

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "login successfully",
    "CoreVersion": "1.0.0",
    "licensetype": "opensource",
    "Rootpath": "/data/gstore",
    "type": "ghttp"
}
```

5.2.5.3 testConnect

Brief description

- Test whether the server can connect (for Workbench)
- Updates in this version: added HTTP service type in the return value

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `httprequest` , passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **testConnect**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
CoreVersion	string	Kernel version
licensetype	string	Certificate type (Open Source or Enterprise)
type	string	HTTP service type (fixed as ghttp)

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "success",
    "CoreVersion": "1.0.0",
    "licensetype": "opensource",
    "type": "ghttp"
}
```

5.2.5.4 getCoreVersion

Brief description

- Get the server version number (for Workbench)

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `httprequest` , passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **getCoreVersion**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
CoreVersion	string	Kernel version
type	string	HTTP service type (fixed as ghttp)

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "success",
    "CoreVersion": "1.0.0",
    "type": "ghttp"
}
```

5.2.5.5 ipmanage

Brief description

- Blacklist and whitelist management

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Querying the blacklist and whitelist:

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **ipmanage**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
type	yes	string	Operation type, fixed value is **1**

Saving the blacklist and whitelist:

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **ipmanage**
username	yes	string	user name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
type	yes	string	Operation type, fixed value is **2**
ip_type	yes	string	List type, 1-Blacklist 2-Whitelist
ips	yes	string	IP List (Multiple separated by **,** supporting range configuration, using **-** connections such as ip1-ip2)

```
//Post example
{
    "operation": "ipmanage",
    "username": "root",
    "password": "123456",
    "type": "2",
    "ip_type":"1",
    "ips": "192.168.1.111,192.168.1.120-192.168.1.129"
}
```

Return value

Parameter	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
ResponseBody	Object	Return data (only for queries)
ip_type	string	List type, 1-Blacklist 2-Whitelist
ips	array	IP List

Return sample

```
//result of querying the black list and white list
{
    "StatusCode": 0,
    "StatusMsg": "success",
    "ResponseBody": {
        "ip_type": "1",
        "ips": [
            "192.168.1.111",
            "192.168.1.120-192.168.1.129"
        ]
    }
}
// result of saving the black list and white list
{
    "StatusCode": 0,
    "StatusMsg": "success"
}
```

5.2.5.6 upload

Brief Description

- Uploading files. Currently supported file formats include nt, ttl, and txt

Request URL

- ghttp service: `http://127.0.0.1:9000/file/upload` (note that the address has changed)
- grpc service: `http://127.0.0.1:9000/grpc/file/upload` (note that the address has changed)

Request method

- POST

Parameter Transfer Methods

- POST request, `form-data` in the `body` of `httprequest` (requires the RequestHeader parameter Content-Type: multipart/form-data

Parameters

Parameter name	Required	Type	Description
username	Yes	string	Username
password	Yes	string	Password (clear text)
encryption	No	string	If it is empty, the password is clear text. If it is 1, it means md5 encryption is used
file	Yes	boudary	Binary file stream of the file to be uploaded

Return Value

Parameter name	Type	Description
StatusCode	int	Return value code value (please refer to the attached table: Return Value Code Table)
StatusMsg	string	Returns specific information
filepath	string	The relative path address returned after successful upload

Return Sample:

```
{
    "StatusCode": 0,
    "StatusMsg": "success",
    "filepath": "./upload/test_20221101164622.nt"
}
```

5.2.5.7 download

Brief Description

- Download files, currently supports downloading files from the gStore root directory.

Request URL

- ghttp service: `http://127.0.0.1:9000/file/download` (Note that the address has changed)
- grpc service: `http://127.0.0.1:9000/grpc/file/upload` (Note that the address has changed)

Request method

- POST

Parameter Transfer Method

- POST request: the parameters are passed in the form of Form data (requires the RequestHeader parameter Content-Type: application/x-www-form-urlencoded)

Parameters

Parameter name	Required	Type	Description
username	Yes	string	Username (the default username is system)
password	Yes	string	Password (clear text)
encryption	No	string	If it is empty, the password is clear text. If it is 1, it means md5 encryption is used
filepath	Yes	string	The path of the file to be downloaded (only supports downloading files from the main directory and subdirectories of gstore)

Return Value

- Binary stream response

Return Sample:

```
Content-Range: bytes 0-389/389
Content-Type: application/octet-stream
Date: Tue, 01 Nov 2022 17:21:40 GMT
Content-Length: 389
Connection: Keep-Alive
```

5.2.5.8 state

Brief Description

- Statistics on system resource information

Request URL

- ghttp service: `http://127.0.0.1:9000/ghttp/api`
- grpc service: `http://127.0.0.1:9000/grpc/api`

Request method

- GET/POST

Parameter Transfer Method

- GET request, the parameters are passed directly as the URL
- POST request: the parameters are passed in the form of Form data (requires the RequestHeader parameter Content-Type: application/x-www-form-urlencoded)

Parameters

Parameter name	Required	Type	Description
operation	yes	string	Operation name, fixed value is **stat**
username	Yes	string	Username (the default username is system)
password	Yes	string	Password (plain text)
encryption	No	string	If it is empty, the password is clear text. If it is 1, it means md5 encryption is used

Return Value

Parameter name	Type	Description
StatusCode	int	Return value code value (please refer to the attached table: Return Value Code Table)
StatusMsg	string	Returns specific information
cup-usage	string	CPU usage ratio
mem_usage	string	Memory usage (in MB)
disk_available	string	Available disk space (in MB)

Return Sample:

```
{
    "StatusCode": 0,
    "StatusMsg": "success",
    "cpu_usage": "10.596026",
    "mem_usage": "2681.507812",
    "disk_available": "12270"
}
```

5.2.5.9 shutdown

Brief description

- close ghttp

Request URL

- ghttp service: `http://127.0.0.1:9000/shutdown` (Note that the address has changed)
- grpc service: `http://127.0.0.1:9000/grpc/shutdown` (Note that the address has changed)

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
username	yes	string	user name（default user name is system)
password	yes	string	Password（This password need to be viewed in the server's system.db/password[port].txt file)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Server stopped successfully."
}
```

5.2.6 Interfaces of System Operation

5.2.6.1 show

Brief description

- Display all database list

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **show**
username	yes	string	user name
password	yes	string	Password (plain text)
			If it is empty, the password is clear text. If it is 1, it

Parameter name	Mandatory	Type	Note
encryption	No	string	If it is empty, the password is clear text. If it is 1, it means it is encrypted with md5.

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
ResponseBody	JSONArray	JSON arrays (each of which is a database information)
database	string	database name
creator	string	creator
built_time	string	create time
status	string	database status

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Get the database list successfully!",
    "ResponseBody": [
        {
            "database": "lubm",
            "creator": "root",
            "built_time": "2021-08-22 11:08:57",
            "status": "loaded"
        },
        {
            "database": "movie",
            "creator": "root",
            "built_time": "2021-08-27 20:56:56",
            "status": "unloaded"
        }
    ]
}
```

5.2.6.2 load

Brief description

- Loading a database into memory is a prerequisite for many operations, such as query and monitor.

Request mode

- GET/POST

Parameter transfer mode

- GET request: the parameters are passed directly as the URL
- POST request: `raw` in `body` in `httprequest` , passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **load**
username	yes	string	User name
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
password	yes	string	Password (plain text)
db_name	yes	string	Database name (.db is not required)
csr	no	string	Whether to load CSR resources, default is 0 (set to 1 when using advanced query functions)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
csr	string	Whether to load CSR resources

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Database loaded successfully.",
    "csr": "1"
}
```

5.2.6.3 unload

Brief description

- Unload the database from memory (all changes are flushed back to hard disk)

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **unload**
db_name	yes	string	Database name (.db is not required)
username	yes	string	user name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Database unloaded."
}
```

5.2.6.4 monitor

Brief description

- Get database statistics (database needs to be loaded)

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **monitor**
username	yes	string	User name
password	yes	string	Password (plain text)

Parameter name	Mandatory	Type	Note
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
db_name	yes	string	Database name (.db is not required)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
database	string	database name
creator	string	creator
builtTime	string	create time
tripleNum	string	number of triples
entityNum	int	number of entities
literalNum	int	number of characters (attribute value)
subjectNum	int	number of subjects
predicateNum	int	number of objects
connectionNum	int	number of connections
diskUsed	int	disk space（MB）
subjectList	Array	Entity type statistics

Return sample


```
{
    "StatusCode": 0,
    "StatusMsg": "success",
    "database": "test_lubm",
    "creator": "root",
    "builtTime": "2021-08-27 21:29:46",
    "tripleNum": "99550",
    "entityNum": 28413,
    "literalNum": 0,
    "subjectNum": 14569,
    "predicateNum": 17,
    "connectionNum": 0,
    "diskUsed": 3024,
    "subjectList": [
        {
            "name": "ub:Lecturer",
            "value": 93
```

```
        },
        {
            "name": "ub:AssistantProfessor",
            "value": 146
        },
        {
            "name": "ub:University",
            "value": 1
        }
    ]
}
```

5.2.6.5 build

Brief description

- Create a database based on existing NT files, or build an empty library (supported in version 1.2)
- The file must exist on the gStore server
- You can first upload data files to the gStore server through the upload interface

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `httprequest` , passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **build**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
db_name	yes	string	database name(no need .db)
db_path	no	string	Database file path, supported file types are (can be absolute path or relative path, relative path is referenced from gStore installation root directory)
async	no	string	If **async is "true"**, immediately return the query opd_id, the server's asynchronous processing logic, and the client can call the checkOperationState interface through opd_id to view the operation status information
callback	no	string	Callback interface, for example: http://127.0.0.1:8080/callback

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
failed_num	int	Number of failed constructions
opt_id	string	Return the operation opd_id, the client can call the checkOperationState interface through opd_id to view the operation status information

Return sample(defult)

```
{
    "StatusCode": 0,
    "StatusMsg": "Import RDF file to database done.",
    "failed_num": 0,
    "opt_id": "XXXX-XXXX-XXXX-XXXX"
}
```

Return sample(async="true")

```
{
    "StatusCode": 0,
    "StatusMsg": "Operation Success.",
    "opt_id": "XXXX-XXXX-XXXX-XXXX"
}
```

Return sample(callback="http://127.0.0.1:8080/callback")

```
#push data to callback address
{
    "operation": "build",
    "StatusCode": 0,
    "StatusMsg": "Import RDF file to database done.",
    "failed_num": 0,
    "opt_id": "XXXX-XXXX-XXXX-XXXX"
}
```

5.2.6.6 drop

Brief description

- Delete the database (either logically or physically)

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **drop**
db_name	yes	string	Database name (.db is not required)
username	yes	string	user name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
is_backup	no	string	True: Logical deletion, false: represents physical deletion (default true)， if it's logical deletion, change the file folder to .bak file foder, user can change the folder name to. db and add the folder to the system database by calling `bin/ ginit-db database name`

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Database test_lubm dropped."
}
```

5.2.6.7 backup

Brief description

- Back up database

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL

- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **backup**
username	yes	string	user name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
db_name	yes	string	database that need operaions
backup_path	no	string	The backup file path can be relative or absolute. The relative path uses the gStore root directory as reference. The default path is the backup directory in the gStore root directory
async	no	string	If **async is "true"**, immediately return the query opd_id, the server's asynchronous processing logic, and the client can call the checkOperationState interface through opd_id to view the operation status information
callback	no	string	Callback interface, for example: http://127.0.0.1:8080/callback

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
backupfilepath	string	Backup file path (this value can be used as the input parameter value for restore)
opt_id	string	Return the operation opd_id, the client can call the checkOperationState interface through opd_id to view the operation status information

Return sample (defult)

```
{
    "StatusCode": 0,
    "StatusMsg": "Database backup successfully.",
    "backupfilepath": "./backups/lubm.db_210828211529",
    "opt_id": "xxxx-xxxx-xxxx-xxxx"
}
```

Return sample (async="true")

```
{
    "StatusCode": 0,
    "StatusMsg": "Operation Success.",
    "opt_id": "xxxx-xxxx-xxxx-xxxx"
}
```

Return sample (callback="http://127.0.0.1:8080/callback")

```
{
    "operation": "backup",
    "StatusCode": 0,
    "StatusMsg": "Database backup successfully.",
    "backupfilepath": "./backups/lubm.db_210828211529",
    "opt_id": "xxxx-xxxx-xxxx-xxxx"
}
```

5.2.6.8 backuppath

Brief description

- Obtain all backup files of the database located under the default backup path.

Request Methods

- GET/POST

Parameter Passing Methods

- For GET requests, parameters are passed directly in the URL.
- For POST requests, the `raw` field in the `body` of the `httprequest` is passed in JSON structure.

Parameters

Parameter Name	Required	Type	Description
operation	Yes	string	Operation name, fixed value is **backuppath**
username	Yes	string	Username
password	Yes	string	Password (plaintext)
encryption	No	string	If it is empty, the password is plaintext. If it is 1, it means encryption with MD5
db_name	Yes	string	Database name that needs to be queried

Return Values

Parameter Name	Type	Description
StatusCode	integer	Return value code (please refer to the attached table for specific values).
StatusMsg	string	Specific return information.
paths	Array	Backup file paths (this value can be used as input for restore operation).

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "success",
    "paths": [
        "./backups/lubm.db_220828211529",
        "./backups/lubm.db_221031094522"
    ]
}
```

5.2.6.9 restore

Brief description

- Restore database

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **restore**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If it is empty, the password is plaintext. If it is 1, it means encryption with MD5.
db_name	yes	string	Database that need operations
backup_path	no	string	The full time-stamped path of the backup file (can be a relative path or an absolute path. The relative path is based on the gStore root directory). The default path is the backup

Parameter name	Mandatory	Type	Note
			directory in the gStore root directory
async	no	string	If **async is "true"**, immediately return the query opd_id, the server's asynchronous processing logic, and the client can call the checkOperationState interface through opd_id to view the operation status information
callback	no	string	Callback interface, for example: http://127.0.0.1:8080/callback

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
opt_id	string	Return the operation opd_id, the client can call the checkOperationState interface through opd_id to view the operation status information

Return sample (defult)

```
{
    "StatusCode": 0,
    "StatusMsg": "Database lumb restore successfully.",
    "opt_id": "XXXX-XXXX-XXXX-XXXX"
}
```

Return sample (async="true")

```
{
    "StatusCode": 0,
    "StatusMsg": "Operation Success.",
    "opt_id": "XXXX-XXXX-XXXX-XXXX"
}
```

Return sample (callback="http://127.0.0.1:8080/callback")

```
{
    "StatusCode": 0,
    "StatusMsg": "Database lumb restore successfully.",
    "opt_id": "XXXX-XXXX-XXXX-XXXX"
}
```

5.2.6.10 query

Brief description

- query the database

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **query**
username	yes	string	user name
password	yes	string	Password (plain text)
encryption	no	string	If it is empty, the password is plaintext. If it is 1, it means encryption with MD5.
db_name	yes	string	database that need operations
format	no	string	The result set returns in json, HTML, and file format. The default is JSOn
sparql	yes	string	Sparql statement to execute (SPARQL requires URL encoding if it is a GET request)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
head	object	Head information
link	array	
vars	array	
results	object	Result information (please refer to the return example for details)
bindings	array	
AnsNum	int	Number of results
OutputLimit	int	Maximum number of return results (-1 for no limit)
ThreadId	string	Query thread ID
QueryTime	string	Query time (milliseconds)

Return sample


```
{
    "head": {
        "link": [],
        "vars": [
            "x"
        ]
    },
    "results": {
        "bindings": [
            {
                "x": {
                    "type": "uri",
                    "value": "House of Flying Daggers"
                }
            },
            {
                "x": {
                    "type": "uri",
                    "value": "Blood Brothers"
                }
            },
            {
                "x": {
                    "type": "uri",
                    "value": "flower"
                }
            }
        ]
    },
    "StatusCode": 0,
    "StatusMsg": "success",
    "AnsNum": 15,
    "OutputLimit": -1,
    "ThreadId": "140595527862016",
    "QueryTime": "1"
}
```


5.2.6.11 export

Brief description

- export database

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **restore**
username	yes	string	user name
password	yes	string	Password (plain text)
encryption	no	string	If it is empty, the password is plaintext. If it is 1, it means encryption with MD5.
db_name	yes	string	database that need operations
db_path	no	string	Export path (gstore root by default)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
filepath	string	Path for exporting files

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Export the database successfully.",
    "filepath": "export/lubm_210828214603.nt"
}
```

5.2.6.12 batchInsert

Brief description

- batch insert data

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **batchInsert**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If it is empty, the password is plaintext. If it is 1, it means encryption with MD5.
db_name	yes	string	Database name
file	no	string	The data to be inserted in the nt file or zip package (which can be a relative path or an absolute path) cannot be empty at the same time as the dir parameter. When both have values, take the file parameter
dir	no	string	The data to be inserted into the nt folder (which can be a relative path or an absolute path) cannot be empty at the same time as the file parameter. When both have values, take the file parameter
async	no	string	If **async is "true"**, immediately return the query opd_id, the server's asynchronous processing logic, and the client can call the checkOperationState interface through opd_id to view the operation status information
callback	no	string	Callback interface, for example: http://127.0.0.1:8080/callback

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
success_num	string	Number of successful executions
failed_num	srting	Number of failed executions
opt_id	string	Return the operation opd_id, the client can call the checkOperationState interface through opd_id to view the operation status information

Return Sample (defult)

```
{
    "StatusCode": 0,
    "StatusMsg": "Batch insert data successfully.",
    "success_num": 25,
    "failed_num": 0,
    "opt_id": "XXXX-XXXX-XXXX-XXXX"
}
```

Return Sample (async="true")

```
{
    "StatusCode": 0,
    "StatusMsg": "Operation Success.",
    "opt_id": "XXXX-XXXX-XXXX-XXXX"
}
```

Return Sample (callback="http://127.0.0.1:8080/callback")

```
#push data to callback address
{
    "operation": "batchInsert",
    "StatusCode": 0,
    "StatusMsg": "Batch insert data successfully.",
    "success_num": 25,
    "failed_num": 0,
    "opt_id": "XXXX-XXXX-XXXX-XXXX"
}
```

5.2.6.13 batchRemove

Brief description

- batch remove data

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **batchRemove**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If it is empty, the password is plaintext. If it is 1, it means encryption with MD5.
db_name	yes	string	Database name

Parameter name	Mandatory	Type	Note
file	no	string	Data NT files to be deleted (can be relative or absolute paths)
async	no	string	If **async is "true"**, immediately return the query opd_id, the server's asynchronous processing logic, and the client can call the checkOperationState interface through opd_id to view the operation status information
callback	no	string	Callback interface, for example: http://127.0.0.1:8080/callback

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
success_num	string	Number of successful executions
opt_id	string	Return the operation opd_id, the client can call the checkOperationState interface through opd_id to view the operation status information

Return Sample (defult)

```
{
    "StatusCode": 0,
    "StatusMsg": "Batch remove data successfully.",
    "success_num": 25,
    "failed_num": 0,
    "opt_id": "xxxx-xxxx-xxxx-xxxx"
}
```

Return Sample (async="true")

```
{
    "StatusCode": 0,
    "StatusMsg": "Operation Success.",
    "opt_id": "xxxx-xxxx-xxxx-xxxx"
}
```

Return Sample (callback="http://127.0.0.1:8080/callback")

```
#push data to callback address
{
    "operation": "batchRemove",
    "StatusCode": 0,
    "StatusMsg": "Batch remove data successfully.",
    "success_num": 25,
    "opt_id": "XXXX-XXXX-XXXX-XXXX"
}
```

5.2.6.14 rename

Brief Description

- Rename the database.

Request method

- GET/POST

Parameter Transfer Method

- For GET requests, parameters are passed directly in the URL.
- For POST requests, the `raw` field in the `body` of the `httprequest` is used to pass parameters in JSON structure.

Parameters

Parameter name	Required	Type	Description
operation	Yes	string	Operation name, the fixed value is **rename**
username	Yes	string	Username
password	Yes	string	Password (clear text)
encryption	No	string	If it is empty, the password is clear text. If it is 1, it means that md5 encryption is used.
db_name	Yes	string	Database name
new_name	Yes	string	New name of the database

Return value

Parameter name	Type	Description
StatusCode	int	Return value code value (please refer to the attached table: Return Value Code Table)
StatusMsg	string	Returns specific information

Return Sample:

```
{
    "StatusCode": 0,
    "StatusMsg": "success"
}
```

5.2.6.15 checkOperationState

Brief description

- Query asynchronous operation status (build, backup, restore, batchInsert, batchRemove)

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **checkOperationState**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If it is empty, the password is plaintext. If it is 1, it means encryption with MD5.
opt_id	yes	string	Operation number

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
state	int	execution state(0 executing , -1 failed execution, 1 successfu execution)
success_num	int	Number of successful executions
failed_num	int	Number of failed executions

Return Sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Batch insert data successfully.",
    "state": 1,
    "success_num": 100,
    "failed_num": 0
}
```

5.2.7 Interface of Database Transaction

5.2.7.1 begin

Brief description

- start transaction

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **begin**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If it is empty, the password is plaintext. If it is 1, it means encryption with MD5.
db_name	yes	string	Database name
isolevel	yes	string	Transaction isolation level 1:RC(read committed) 2:SI(snapshot isolation) 3:SR(serializable)

			2.3I(snapshot isolation) 3.3R(serializable)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
TID	string	Transaction ID(this ID is important enough to take as a parameter

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "transaction begin success",
    "TID": "1"
}
```

5.2.7.2 tquery

Brief description

- query the transaction type

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **tquery**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If it is empty, the password is plaintext. If it is 1, it means encryption with MD5.
db_name	yes	string	Database name
tid	yes	string	Transaction ID
sparql	yes	string	sparql statement

Return value

Parameter Name	Type	Description
StatusCode	int	Return value code (please refer to the attached table for specific values)
StatusMsg	string	Specific return information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "success"
}
```

5.2.7.3 commit

Brief description

- submit transaction

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **commit**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If it is empty, the password is plaintext. If it is 1, it means encryption with MD5.
db_name	yes	string	Database name
tid	yes	string	Transaction ID

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "transaction commit success. TID: 1"
}
```

5.2.7.4 rollback

Brief description

- Rollback trasnsaction

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **rollback**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If it is empty, the password is plaintext. If it is 1, it means encryption with MD5.
db_name	yes	string	Database name
tid	yes	string	Transaction ID

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "transaction rollback success. TID: 2"
}
```

5.2.7.5 checkpoint

Brief description

- Received Flush data back to hard disk (to make data final)

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **checkpoint**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If it is empty, the password is plaintext. If it is 1, it means encryption with MD5.
db_name	yes	string	Database name

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Database saved successfully."
}
```

5.2.8 Interfaces of User Management

5.2.8.1 showuser

Brief description

- Display all user information

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **showuser**
username	yes	string	user name
password	yes	string	Password (plain text)
encryption	No	string	If it is empty, the password is clear text. If it is 1, it means it is encrypted with md5.

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
ResponseBody	JsonArray	JSON object array
username	string	user name
password	string	password
query_privilege	string	Query permissions (database names separated by commas)
update_privilege	string	Update permissions (database names separated by commas)
load_privilege	string	Load permissions (database names separated by commas)

Parameter name	Type	Note
unload_privilege	string	Unload permissions (database names separated by commas)
backup_privilege	string	Back up permissions (database names separated by commas)
restore_privilege	string	Restore permissions (database names separated by commas)
export_privilege	string	Export permissions (database names separated by commas)

Return sample


```
{
    "StatusCode": 0,
    "StatusMsg": "success",
    "ResponseBody": [
        {
            "username": "liwenjie",
            "password": "shuaige1982",
            "query_privilege": "",
            "update_privilege": "",
            "load_privilege": "",
            "unload_privilege": "",
            "backup_privilege": "",
            "restore_privilege": "",
            "export_privilege": ""
        },
        {
            "username": "root",
            "password": "123456",
            "query_privilege": "all",
            "update_privilege": "all",
            "load_privilege": "all",
            "unload_privilege": "all",
            "backup_privilege": "all",
            "restore_privilege": "all",
            "export_privilege": "all"
        }
    ]
}
```


5.2.8.2 usermanage

Brief description

- Manage users (including adding, deleting, and changing users)

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **usermanage**
type	yes	string	Operation Type（1：adduser，2：deleteUser 3：alterUserPassword）
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	No	string	If it is empty, the password is clear text. If it is 1, it means it is encrypted with md5.
op_username	yes	string	User name for the operation
op_password	yes	string	Password of the operation (if the password is to be changed, the password is the password to be changed) (If the operation contains special characters and the get request is adopted, the value must be urlencode-encoded)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 1004,
    "StatusMsg": "username already existed, add user failed."
}
```

Notes

- The default interface permissions for the new user are: `login`, `check`, `testConnect`, `getCoreVersion`, `show`, `funquery`, `funcudb`, `funreview`, `userpassword`.
- Users with `query` rights also have the following interface rights: `query`, `monitor`.
- Users with the `update` permission also have the following interface permissions: `batchInsert`, `batchRemove`, `begin`, `tquery`, `commit`, `rollback`.
- Only the root user can invoke the interface rights that are not within the scope of authorization management, for example: `build`, `drop`, `usermanage`, `showuser`, `userprivilege`, `manage`, `txnlog`, `checkpoint`, `shutdown`, `querylog`, `accesslog`, `ipmanage`.

5.2.8.3 userprivilegemanage

Brief description

- Manage user permissions (including adding, deleting, and changing users)

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **userprivilegemanage**
type	yes	string	Operation type（1：add privilege，2：delete privilege 3：clear Privilege ）
username	yes	string	user name
password	yes	string	Password (plain text)
encryption	No	string	If it is empty, the password is clear text. If it is 1, it means it is encrypted with md5.
op_username	yes	string	User name for the operation
privileges	no	string	Permissions to operate on (multiple permissions separated by commas) (can be null for clear Privilege)1:query,2:load,3:unload,4:update,5:backup,6:restore,7:export, you can set multi privileges by using , to split.
db_name	no	string	The database to operate on (this can be empty if it is clearPrivilege

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "add privilege query successfully. \r\nadd privilege load
successfully. \r\nadd privilege unload successfully. \r\nadd privilege update
successfully. \r\nadd privilege backup successfully. \r\n"
}
```

5.2.8.4 userpassword

Brief description

- Modify the user password.

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `httprequest` , passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **userpassword**
username	yes	string	User name
password	yes	string	Password
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
op_password	yes	string	New password (plain text)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode":0,
    "StatusMsg": "change password done."
}
```

5.2.9 Interface of Custom Function

5.2.9.1 funquery

Brief description

- Show user-defined graph analysis functions

Request mode

- POST

Parameter transfer mode

- POST request, `raw` in `body` in `httprequest` , passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **funquery**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
funInfo	no	JSONObject	Query parameters
funName	no	string	Function name
funStatus	no	string	Status (1- to compile; 2- compiled; 3- exception)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
list	JSONArray	JSON array (if there is no data, no array is returned)
funName	string	Name
funDesc	string	Description
funArgs	string	Parameter type (1- no K hop parameter; 2- with K hop parameter)
funBody	string	Function content
funSubs	string	Child functions (can be called in funBody)
funStatus	string	Status (1- to compile; 2- compiled; 3- exception)
lastTime	string	Last edit time (yyyy-MM-dd HH:mm:ss)

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "success",
    "list": [
        {
            "funName": "demo",
            "funDesc": "this is demo",
            "funArgs": "2",
```

```
            "funBody": "{\nstd::cout<<\"uid=\<<uid<<endl;\nstd::cout<<\"vid=\
<<vid<<endl;\nstd::cout<<\"k=\"<<k<<endl;\nreturn \"success\";\n}",
            "funSubs": "",
            "funStatus": "1",
            "lastTime": "2022-03-15 11:32:25"
        }
    ]
}
```


5.2.9.2 funcudb

Brief description

- User-defined graph analysis function management (add, modify, delete, compile)

Request mode

- POST

Parameter transfer mode

- POST request, `raw` in `body` in `httprequest` , passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **funcudb**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
type	yes	string	1: add, 2: modify, 3: delete, 4: compile
funInfo	yes	JSONObject	Operator function
funName	yes	string	Function name
funDesc	no	string	Description
funArgs	no	string	Parameter type (1- no K hop parameter; 2- with K hop parameter).

Parameter name	Mandatory	Type	Note
funBody	no	string	Function contents (wrapped in {}).
funSubs	no	string	Child functions (can be called in funBody)
funReturn	no	string	Return type(decode is required)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "success"
}
```

5.2.9.3 funreview

Brief description

- Preview the user-defined graph analysis function

Request mode

- POST

Parameter transfer mode

- POST request, `raw` in `body` in `httprequest` , passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **funreview**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
funInfo	yes	JSONObject	Operator function

Parameter name	Mandatory	Type	Note
funName	yes	string	Function name
funDesc	no	string	Description
funArgs	yes	string	Parameter type (1- no K hop parameter; 2- with K hop parameter)
funBody	yes	string	Function contents (contents wrapped in {})
funSubs	yes	string	Child function (can be called in funBody)
funReturn	yes	string	Return type (`path`: the path class result; `value`: value class result)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
result	string	Function source code (decode is required)

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "success",

 "Result":"%23include+%3Ciostream%3E%0A%23include+%22..%2F..%2FDatabase%2FCSRUtil.h%22%0A%0Ausing+..."
}
```

5.2.10 Interface of Log

5.2.10.1 txnlog

Brief description

- Get transaction logs (this function is only available for the root user)
- Updates in this version: added paging query parameters

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `httprequest` , passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **txnlog**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
pageNo	yes	int	Page number, the value ranges from 1 to N, the default value is 1
pageSize	yes	int	The number of entries per page, the value ranges from 1 to N, the default value is 10

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
totalSize	int	Total size
totalPage	int	The total number of pages
pageNo	int	The current page number
pageSize	int	The number of entries per page
list	JSONArray	Log array
db_name	string	Database name
TID	string	Transaction ID
user	string	User

Parameter name	Type	Note
state	string	State (COMMITED/RUNNING/ROLLBACK)
begin_time	string	Start time
end_time	string	End time

Return sample


```
{
    "StatusCode": 0,
    "StatusMsg": "Get Transaction log success",
    "totalSize": 2,
    "totalPage": 1,
    "pageNo": 1,
    "pageSize": 10,
    "list": [
        {
            "db_name": "lubm2",
            "TID": "1",
            "user": "root",
            "begin_time": "1630376221590",
            "state": "COMMITED",
            "end_time": "1630376277349"
        },
        {
            "db_name": "lubm2",
            "TID": "2",
            "user": "root",
            "begin_time": "1630376355226",
            "state": "ROLLBACK",
            "end_time": "1630376379508"
        }
    ]
}
```


5.2.10.2 querylogdate

Brief description

- Get the date of gStore's query log (for the date parameter of the query log interface)

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **querylogdate**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5

Return value

Parameter	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
list	array	The list of dates

Return sample

```
{
    "StatusCode":0,
    "StatusMsg":"Get query log date success",
    "list":[
        "20220828",
        "20220826",
        "20220825",
        "20220820"
    ]
}
```

5.2.10.3 querylog

Brief description

- Obtaining query Logs

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **querylog**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
date	yes	string	Date format is yyyyMMdd
pageNo	yes	int	Page number. The value ranges from 1 to N. The default value is 1
pageSize	yes	int	The number of entries per page. The value ranges from 1 to N. The default value is 10

Return value

Parameter	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
totalSize	int	Total number
totalPage	int	Total page number
pageNo	int	Current page number
pageSize	int	The number of entries per page
list	Array	Log array
QueryDateTime	string	Query date/time
Sparql	string	SPARQL statement
Format	string	Query return format
RemoteIP	string	Request IP
FileName	string	Query result set files
QueryTime	int	Time (ms)
AnsNum	int	Result number

Return sample

```
{
    "StatusCode":0,
```

```
    "StatusMsg":"Get query log success",
    "totalSize":64,
    "totalPage":13,
    "pageNo":2,
    "pageSize":5,
    "list":[
        {
            "QueryDateTime":"2021-11-16 14:55:52:90ms:467microseconds",
            "Sparql":"select ?name where { ?name <不喜欢> <Eve>. }",
            "Format":"json",
            "RemoteIP":"183.67.4.126",
            "FileName":"140163774674688_20211116145552_847890509.txt",
            "QueryTime":0,
            "AnsNum":2
        }
        ......
    ]
}
```

5.2.10.4 accesslogdate

Brief description

- Get the date of gStore's access log (for the date parameter of the access log interface)

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **querylogdate**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5

Return value

Parameter	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
list	array	The list of dates

Return sample

```
{
    "StatusCode":0,
    "StatusMsg":"Get access log date success",
    "list":[
        "20220913",
        "20220912",
        "20220911",
        "20220818",
        "20220731",
        "20220712",
        "20220620",
    ]
}
```

5.2.10.5 accesslog

Brief description

- Get the access log of the API

Request mode

- GET/POST

Parameter transfer mode

- GET request, the parameters are passed directly as the URL
- POST request, `raw` in `body` in `Httprequest`, passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **accesslog**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
date	yes	string	Date format is yyyyMMdd
pageNo	yes	int	Page number. The value ranges from 1 to N. The default value is 1
pageSize	yes	int	The number of entries per page. The value ranges from 1 to N. The default value is 10

Return value

Parameter	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
totalSize	int	Total number
totalPage	int	Total page number
pageNo	int	Current page number
pageSize	int	The number of entries per page
list	Array	Log array
ip	string	The IP that makes the access
operation	string	The type of operation performed
createtime	string	The time of the operation
code	string	The result of the operation (refer to Appendix: Return value code table for details)
msg	string	Log description

Return sample

```
{
    "StatusCode":0,
    "StatusMsg":"Get access log success",
    "totalSize":64,
    "totalPage":13,
    "pageNo":2,
    "pageSize":5,
    "list":[
        {
            "ip":"127.0.0.1",
            "operation":"StopServer",
            "createtime":"2021-12-14 09:55:16",
            "code":0,
            "msg":"Server stopped successfully."
        }
        ......
    ]
}
```

5.2.11 Reason Engine

There are 7 types of operation to parameter type

- 1:addReason
- 2:listReason
- 3:compileReason
- 4:executeReason

- 5:disableReason
- 6.showReason
- 7.deleteReason

5.2.11.1 addReason

Brief description

- add reasoning rules

Request mode

- POST

Parameter transfer mode

- POST request, `raw` in `body` in `Httprequest`, passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **reasonManage**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
db_name	yes	string	Database name
type	yes	string	Operation type is "1"

Parameter name	Mandatory	Type	Note
ruleinfo	yes	object	Rule information: "ruleinfo": {"rulename":"", ...}, Rule details
rulename	yes	string	Rule name (Same database ,unique rulename)
description	no	string	Rule description
isenable	yes	int	Whether to enable, optional values: 1 enable ,0 disable
type	yes	int	Reason type:optional values:1 relational reason , 0 attribute reason
logic	yes	int	The relationship between conditions and conditions, optional values: 1 logical and , 0 logical or
conditions	yes	array	Rules set
patterns	yes	array	Triples set
subject	yes	string	Subject
predicate	yes	string	Predicate
object	yes	string	Object
filters	no	array	Filter conditions
count_info	no	Object	Aggregate function (undetermined)
return	yes	Object	Return object
source	yes	string	Source node
target	no	string	Terminal node
label	否	string	predicate (The system automatically adds the Rule: prefix)
value	否	string	Attribute value

Return value

Parameter	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Relational reasoning rule sample

```
{
    "rulename": "uncle",
```

```
    "description": "The mother's male sibling was the uncle"
    "isenable": 1,
    "type": 1,
    "logic": 1,
    "conditions": [
        {
            "patterns": [
                {
                    "subject": "?x ",
                    "predicate": "<mother>",
                    "object": "?y"
                },
                {
                    "subject": "?y ",
                    "predicate": "<father>",
                    "object": "?z"
                },
                {
                    "subject": "?k",
                    "predicate": "<father>",
                    "object": "?z"
                },
                {
                    "subject": "?k",
                    "predicate": "<gender>",
                    "object":
"\"male\"^^<http://www.w3.org/2001/XMLSchema#string>"
                }
            ],
            "filters": [],
            "count_info": {}
        }
    ],
    "return": {
        "source": "?x",
        "target": "?k",
        "label": "uncle"
    }
}
```

Attribute reasoning rule sample

```
{
    "rulename": "orphan",
    "description": "Both parents died and were orphaned",
    "isenable": 1,
    "type": 0,
    "logic": 1,
    "conditions": [
        {
            "patterns": [
                {
                    "subject": "?x ",
                    "predicate": "<father>",
                    "object": "?y"
                },
                {
```

```
                    "subject": "?x",
                    "predicate": "<mother>",
                    "object": "?z"
                },
                {
                    "subject": "?y",
                    "predicate": "<state>",
                    "object":
"\"died\"^^<http://www.w3.org/2001/XMLSchema#string>"
                },
                {
                    "subject": "?z",
                    "predicate": "<state>",
                    "object":
"\"died\"^^<http://www.w3.org/2001/XMLSchema#string>"
                }
            ],
            "filters": [],
            "count_info": {}
        }
    ],
    "return": {
        "source": "?x",
        "label": "died",
        "value": "\"orphan\"^^<http://www.w3.org/2001/XMLSchema#string>"
    }
}
```

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Save Successfully! the file path is
./dbhome/lubm.db/reason_rule_files/test.json"
}
```

5.2.11.2 listReason

Brief description

- show rules list

Request mode

- POST

Parameter transfer mode

- POST request, `raw` in `body` in `Httprequest`, passed as `JSON`

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **reasonManage**
username	yes	string	Username
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
db_name	yes	string	Database name
type	yes	string	Operation type：“2”

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
num	int	Rule number
list	JSONArray	Rule name list
rulename	string	Rule name（Same database，unique rule name）
description	string	Rule description
isenable	int	Whether to enable, optional values: 1 enable ,0 disable
type	int	Reason type:optional values:1 relational reason , 0 attribute reason
logic	int	The relationship between conditions and conditions，optional values：1 logical and , 0 logical or
conditions	array	Rules set

Parameter name	Type	Note
condition	arry	Rule
patterns	array	Triples set
subject	string	Subject
predicate	string	Predicate
object	string	Object
filters	array	Filter rule
count_info	Object	Aggregate function（undetermined）
return	Object	Return object
source	string	Source node
target	string	Terminal node
label	string	predicate（The system automatically adds the Rule: prefix）
value	string	Attribute value
status	string	status
insert_sparql	string	insert sparql
delete_sparql	string	delete sparql
createtime	string	create time

Return value


```
{
    "StatusCode": 0,
    "StatusMsg": "ok",
    "list": [
        {
            "ruleid": "002",
            "rulename": "orphan",
            "description": "Both parents died and were orphaned",
            "conditions": [
                {
                    "condition": {
                        "patterns": [
                            {
                                "subject": "?x ",
                                "predicate": "<father>",
                                "object": "?y"
                            },
                            {
                                "subject": "?x",
                                "predicate": "<mother>",
                                "object": "?z"
```

```
                            },
                            {
                                "subject": "?y",
                                "predicate": "<state>",
                                "object":
"\"died\"^^<http://www.w3.org/2001/XMLSchema#string>"
                            },
                            {
                                "subject": "?z",
                                "predicate": "<state>",
                                "object":
"\"died\"^^<http://www.w3.org/2001/XMLSchema#string>"
                            }
                        ],
                        "filters": [],
                        "count_info": {}
                    },
                    "logic": 1
                }
            ],
            "isenable": 1,
            "type": 0,
            "return": {
                "source": "?x",
                "label": "state",
                "value": "\"orphan\"^^<http://www.w3.org/2001/XMLSchema#string>"
            },
            "createtime": "2023-12-29 16:58:00",
            "status": "executed",
            "insert_sparql": "insert {?x <Rule:state>
\"orphan\"^^<http://www.w3.org/2001/XMLSchema#string>. } where {  ?x  <father> ?
y. ?x <mother> ?z. ?y <state>
\"died\"^^<http://www.w3.org/2001/XMLSchema#string>. ?z <state>
\"died\"^^<http://www.w3.org/2001/XMLSchema#string>. }",
            "delete_sparql": "delete where {?x <Rule:state>
\"orphan\"^^<http://www.w3.org/2001/XMLSchema#string>.}"
        },
        {
            "ruleid": "001",
            "rulename": "uncle2",
            "description": "The mother's male sibling was the uncle",
            "conditions": [
                {
                    "condition": {
                        "patterns": [
                            {
                                "subject": "?x ",
                                "predicate": "<mother>",
                                "object": "?y"
                            },
                            {
                                "subject": "?y ",
                                "predicate": "<father>",
                                "object": "?z"
                            },
                            {
                                "subject": "?k",
                                "predicate": "<father>",
```

```
                                "object": "?z"
                            },
                            {
                                "subject": "?k",
                                "predicate": "<gender>",
                                "object":
"\"male\"^^<http://www.w3.org/2001/XMLSchema#string>"
                            }
                        ],
                        "filters": [],
                        "count_info": {}
                    },
                    "logic": 1
                }
            ],
            "isenable": 1,
            "type": 1,
            "return": {
                "source": "?x",
                "target": "?k",
                "label": "uncle",
                "value": ""
            },
            "createtime": "2023-12-29 16:58:00",
            "status": "executed",
            "insert_sparql": "insert { ?x <Rule:uncle> ?k. } where {  ?x
<mother> ?y. ?y  <father> ?z. ?k <father> ?z. ?k <gender>
\"male\"^^<http://www.w3.org/2001/XMLSchema#string>. }",
            "delete_sparql": "delete where {?x <Rule:unlce> ?y.}"
        }
    ],
    "num": 2
}
```


5.2.11.3 compileReason

Brief description

- compile rules

Request mode

- POST

Parameter transfer mode

- POST request: `raw` in `body` in `httprequest` , passed as `JSON`

Parameter

parameter name	Mandatory	Type	Note
operation	yes	string	Operation name.fixed **reasonManage**
username	yes	string	User name
password	yes	string	Password (plain text)
			If the value is null, the password is in plain text. If

parameter name	Mandatory	Type	Note
encryption	no	string	the value is 1, the password is encrypted using md5
db_name	yes	string	Database name
type	yes	string	Operation type：“3”
rulename	yes	string	Rule name

Return value

parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
insert_sparql	string	insert sparql
delete_sparql	string	delete sparql

Return value

```
{
    "insert_sparql": "insert {?x <Rule:state>
\"orphan\"^^<http://www.w3.org/2001/XMLSchema#string>. } where {  ?x  <father> ?
y. ?x <mother> ?z. ?y <state>
\"died\"^^<http://www.w3.org/2001/XMLSchema#string>. ?z <state>
\"died\"^^<http://www.w3.org/2001/XMLSchema#string>. }",
    "delete_sparql": "delete where {?x <Rule:state>
\"orphan\"^^<http://www.w3.org/2001/XMLSchema#string>.}",
    "StatusCode": 0,
    "StatusMsg": "ok"
}
```

5.2.11.4 executeReason

Brief description

- execute rules

Request mode

- POST

Parameter transfer mode

- POST request, `raw` in `body` in `Httprequest`, passed as `JSON`

Parameter

parameter	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **reasonManage**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
db_name	yes	string	Database name
type	yes	string	Operation type：“4”
rulename	yes	string	Rule name

Return value

parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
insert_sparql	string	insert sparql
AnsNum	int	successful answer number

return value

```
{
    "insert_sparql": "insert {?x <Rule:state>
\"orphan\"^^<http://www.w3.org/2001/XMLSchema#string>. } where {  ?x  <father> ?
y. ?x <mother> ?z. ?y <state>
\"died\"^^<http://www.w3.org/2001/XMLSchema#string>. ?z <state>
\"died\"^^<http://www.w3.org/2001/XMLSchema#string>. }",
    "AnsNum": 0,
    "StatusCode": 0,
    "StatusMsg": "ok"
}
```

5.2.11.5 disableReason

Brief description

- Invalidate the rule

Request mode

- POST

Parameter transfer mode

- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed **reasonManage**
username	yes	string	User name
password	yes	string	Password (plain text)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
db_name	yes	string	Database name
type	yes	string	Operation type：“5”
rulename	yes	string	Rule name

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
delete_sparql	string	Delete sparql
AnsNum	int	successful answer number

Return value

```
{
    "delete_sparql": "delete where {?x <Rule:state>
\"orphan\"^^<http://www.w3.org/2001/XMLSchema#string>.}",
    "AnsNum": 1,
    "StatusCode": 0,
    "StatusMsg": "ok"
}
```

5.2.11.6 showReason

Brief description

- show rule details

Request mode

- POST

Parameter transfer mode

- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value **reasonManage**
username	yes	string	User name
password	yes	string	Password（plain test)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
db_name	yes	string	Database name
type	yes	string	Operation type：“6”
rulename	yes	string	Rule name

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
ruleinfo	Object	Object
insert_sparql	string	insert sparql
delete_sparql	string	delete sparql
status	string	status
rulename	string	Rule name（Same database，unique rule name）
description	string	Rule description
isenable	int	Whether to enable
type	int	1：relational reasoning 0：attribute reasoning
logic	int	Relationship between conditions and conditions. Optional values：1: logical and ; 0 : logical or
createtime	string	Create time

Parameter name	Type	Note
conditions	JSONArray	Rule conditions（Array）
condition	Object	Rule
patterns	array	Triples set
subject	string	Subject
predicate	string	Predicate
object	string	Object
filters	array	Filter conditions
count_info	Object	Aggregate function （undermined）
return	Object	Return object
source	string	Source node
target	string	Terminal node
label	string	predicate（The system automatically adds the Rule: prefix）
value	string	Attribute value

Return value

```
{
    "ruleid": "002",
    "rulename": "orphan",
    "description": "The person whose both parents died is orphan",
    "conditions": [
        {
            "condition": {
                "patterns": [
                    {
                        "subject": "?x ",
                        "predicate": "<father>",
                        "object": "?y"
                    },
                    {
                        "subject": "?x",
                        "predicate": "<mother>",
                        "object": "?z"
                    },
                    {
                        "subject": "?y",
                        "predicate": "<state>",
                        "object":
"\"died\"^^<http://www.w3.org/2001/XMLSchema#string>"
                    },
                    {
                        "subject": "?z",
                        "predicate": "<state>",
                        "object":
"\"died\"^^<http://www.w3.org/2001/XMLSchema#string>"
```

```
                    }
                ],
                "filters": [],
                "count_info": {}
            },
            "logic": 1
        }
    ],
    "isenable": 1,
    "type": 0,
    "return": {
        "source": "?x",
        "label": "state",
        "value": "\"orphan\"^^<http://www.w3.org/2001/XMLSchema#string>"
    },
    "createtime": "2023-12-29 16:58:00",
    "status": "disabled",
    "insert_sparql": "insert {?x <Rule:state>
\"orphan\"^^<http://www.w3.org/2001/XMLSchema#string>. } where {  ?x  <father> ?
y. ?x <mother> ?z. ?y <state>
\"died\"^^<http://www.w3.org/2001/XMLSchema#string>. ?z <state>
\"died\"^^<http://www.w3.org/2001/XMLSchema#string>. }",
    "delete_sparql": "delete where {?x <Rule:state>
\"orphan\"^^<http://www.w3.org/2001/XMLSchema#string>.}",
    "StatusCode": 0,
    "StatusMsg": "ok"
}
```


5.2.11.7 deleteReason

Brief description

- Delete rule

Request mode

- POST

Parameter transfer mode

- POST request, `raw` in `body` in `Httprequest`, passed as `JSON` structure

Parameter

Parameter	Mandatory	Type	Note
operation	yes	string	Operation name, fixed value is **reasonManage**
username	yes	string	User name
password	yes	string	Password（plain test)
encryption	no	string	If the value is null, the password is in plain text. If the value is 1, the password is encrypted using md5
db_name	yes	string	Database name
type	yes	string	Operation type：“7”
rulename	yes	string	Rule name

rulename	yes	string	Rule name

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return value

```
{
    "StatusCode": 0,
    "StatusMsg": "the reason file has been remove successfully! file
path:./dbhome/reason.db/reason_rule_files/orphan.json"
}
```

5.2.12 Appendix 1: Return value code table for details

Code	Details
0	Success
1000	The method type is not support
1001	Authentication Failed
1002	Check Privilege Failed
1003	Param is illegal
1004	The operation conditions are not satisfied
1005	Operation failed
1006	Add privilege Failed
1007	Loss of lock
1008	Transcation manage Failed
1100	The operation is not defined
1101	IP Blocked

5.3 Socket API Instruction

5.3.1 API Interconnection Mode

The gServer interface uses the `socket` protocol and supports multiple ways to access the interface. If the port `9000` is started from the gServer in the Main directory, the contents of the interface interconnection are as follows:

API address:

```
http://ip:9000/
```

The API supports the input of a parameter list in JSON format, as shown below:

```
{"op": "[op_type]", "[paramname1]": "[paramvalue1]", "[paramname2]": "
[paramvalue2]"……}
```

5.3.2 API List

API name	Definition	Note
build	Build graph database	The database file must be locally stored on the server
load	Load graph database	Load the database into memory
unload	Unload graph database	Unload the database from memory
drop	Delete graph database	Logical deletion and physical deletion can be performed
show	Display graph database	Display list of all databases
query	Query graph database	Including query, delete, and insert
stop	Close server	Only root user root can perform this operation
close	Close client server connection	Process client connection closure requests
login	login to database	authenticate user names and password

5.3.3 API Specific Instruction

> This section describes the input and output parameters of each interface. Assume that the IP address of the gserver is 127.0.0.1 and the port is 9000

(1) build- build database

Brief description

- Create a database based on existing NT file
- Files must exist on the gStore server

Request ip

- `127.0.0.1`

Request port number

- `9000`

Parameter transfer mode

- Pass it as a `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
op	yes	string	Operation name, fixed value is **build**
db_name	yes	string	Database name (.db is not required)
db_path	yes	string	Database file path (can be an absolute path or a relative path. The relative path uses the gStore installation root directory as a reference directory)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Import RDF file to database done."
}
```

(2) load

Brief description

- To load a database into memory, a load operation is a prerequisite for many operations, such as Query

Request ip

- `127.0.0.1`

Request port number

- `9000`

Parameter transfer mode

- Pass it as a `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
op	yes	string	Operation name, fixed value is **load**
db_name	yes	string	Database name (.db is not required)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Load database successfully."
}
```

(3) unload

Brief description

- Unmount the database from memory (all changes are flushed back to hard disk)

Request ip

- `127.0.0.1`

Request port number

- `9000`

Parameter transfer mode

- Pass it as a `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
op	yes	string	Operation name, fixed value is **unload**
db_name	yes	string	Database name (.db is not required)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Unload database done."
}
```

(4) drop

Brief description

- Delete the database

Request ip

- `127.0.0.1`

Request port number

- `9000`

Parameter transfer mode

- Pass it as a `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
op	yes	string	Operation name, fixed value is **drop**
db_name	yes	string	Database name (.db is not required)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Drop database done."
}
```

(5) show

Brief description

- Display all database list

Request ip

- `127.0.0.1`

Request port number

- `9000`

Parameter transfer mode

- Pass it as a `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
op	yes	string	Operation name, fixed value is **show**

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
ResponseBody	JSONArray	JSON arrays (each of which is a database information0
-------- database	string	database name
---------status	string	database status

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "success",
    "ResponseBody": [
           "lubm": "loaded",
           "lubm10K": "unloaded"
    ]
}
```

(6) query

Brief description

- query database

Request ip

- `127.0.0.1`

Request port number

- `9000`

Parameter transfer mode

- Pass it as a `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
op	yes	string	Operation name, fixed value is **query**
db_name	yes	string	database that need operations
format	no	string	The result set return format is json by default
sparql	yes	string	The SPARQL statement to execute

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information
head	JSON	Head information
results	JSON	Result information (see Return Sample for details)

Return sample


```
{
    "head": {
        "link": [],
        "vars": [
            "x"
        ]
    },
    "results": {
        "bindings": [
            {
                "x": {
                    "type": "uri",
                    "value": "十面埋伏"
                }
            },
            {
                "x": {
                    "type": "uri",
                    "value": "投名状"
                }
            },
            {
                "x": {
                    "type": "uri",
                    "value": "如花"
```

```
                }
            }
        ]
    },
    "StatusCode": 0,
    "StatusMsg": "success"
}
```

(7) login

Brief description

- Login user (verify username and password)

Request ip

- `127.0.0.1`

Request port number

- `9000`

Parameter transfer mode

- Pass it as a `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
op	yes	string	Operation name, fixed value is **login**
username	yes	string	user name
password	yes	string	Password (plain text)

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 1001,
    "StatusMsg": "wrong password."
}
```

(8) stop

Brief description

- Close server

Request ip

- `127.0.0.1`

Request port number

- `9000`

Parameter transfer mode

- Pass it as a `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
op	yes	string	Operation name, fixed value is **stop**

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Server stopped."
}
```

(9) close

Brief description

- Close the connection to the client

Request ip

- `127.0.0.1`

Request port number

- `9000`

Parameter transfer mode

- Pass it as a `JSON` structure

Parameter

Parameter name	Mandatory	Type	Note
op	yes	string	Operation name, fixed value is **close**

Return value

Parameter name	Type	Note
StatusCode	int	Return value code value (refer to Appendix: Return value code table for details)
StatusMsg	string	Return specific information

Return sample

```
{
    "StatusCode": 0,
    "StatusMsg": "Connection disconnected."
}
```

Appendix 1 return value code table

Code value	Definition
0	Success
1000	The method type is not support
1001	Authentication Failed
1002	Check Privilege Failed
1003	Param is illegal
1004	The operation conditions are not satisfied
1005	Operation failed
1006	Add privilege Failed
1007	Loss of lock
1008	Transcation manage Failed
1100	The operation is not defined
1101	IP Blocked

5.4 C++ HTTP API

To use the C++ API, put the phrase `'#include "client.h"` in your CPP code, as shown below:

Construct the initialization function

```
GstoreConnector(std::string serverIP, int serverPort, std::string httpType,
std::string username, std::string password);
Function: Initialize
Parameter Definition: [Server IP], [HTTP port on the server], [HTTP service
type], [Username], [password]
Example: GstoreConnector gc("127.0.0.1", 9000, "ghttp", "root", "123456");
```

Build database: build

```
std::string build(std::string db_name, std::string rdf_file_path, std::string
request_type);
Function: Create a new database from an RDF file
Parameter Definition: [database name], [.nt file path], [request type "GET" and
"POST ", if request type "GET" can be omitted]
Example: gc.build("lubm", "data/lubm/lubm.nt");
```

Load database: load

```
std::string load(std::string db_name, std::string request_type);
Function：Load the database you created
Parameter Definition：[database name], [request type "GET" and "POST ", if
request type "GET" can be omitted]
Example：gc.load("lubm");
```

Stopping database loading: unload

```
std::string unload(std::string db_name, std::string request_type);
Function：Stopping database loading
Parameter Definition：[database name], [request type "GET" and "POST ", if
request type "GET" can be omitted]
Example：gc.unload("lubm");
```

User management: user

```
std::string user(std::string type, std::string username2, std::string addition,
std::string request_type);
Function：The root user can add, delete, or modify the user's permission only
1.Add or delete users：
Parameter Definition：["add_user" adds user, "delete_user" deletes user],[user
name],[password],[request type "GET" and "POST ", if request type "GET" can be
omitted]
Example：gc.user("add_user", "user1", "111111");
2.Modify user's privilege：
Parameter Definition：["Add_query" adds query permission, "delete_query" deletes
query permission, "add_load" adds load permission, "delete_load" deletes load
permission, "add_unload" adds no load permission, "delete_unload" deletes no
load permission,  "Add_update" adds update permission, "delete_update" deletes
update permission, "add_backup" adds backup permission, "delete_bakup" deletes
backup permission, "add_restore" adds restore permission,  "Delete_restore"
deletes restore permission,"add_export" adds export permission," delete_export"
deletes export permission],[user name],[database name],[Request type "GET" and
"POST ", if the request type is "GET", it can be omitted.]
Example：gc.user("add_query", "user1", "lubm");
```

Display user: showUser

```
std::string showUser(std::string request_type);
Function：Display all Users
Parameter Definition：[Request types "GET" and "POST" can be omitted if the
request type is "GET"]
Example：gc.showUser();
```

Database query: query

```
std::string query(std::string db_name, std::string format, std::string sparql,
std::string request_type);
Function：Query the database
Parameter Definition：[database name], [query result type JSON, HTML or text],
[SPARQL statement], [Request type "GET" and "POST ", if the request type is
"GET", it can be omitted]
Example：
std::string res = gc.query("lubm", "json", sparql);
std::cout << res << std::endl;  //output result
```

Deleting a Database: drop

```
std::string drop(std::string db_name, bool is_backup, std::string request_type);
Function：Delete the database directly or delete the database while leaving a
backup
Parameter Definition：[database name], [false not backup, true backup], [request
type "GET" and "POST ", if request type "GET" can be omitted]
Example：gc.drop("lubm", false);  //Delete the database without leaving a backup
```

Monitoring database: monitor

```
std::string monitor(std::string db_name, std::string request_type);Function：
Displays information for a specific database.
Parameter Definition：[database name], [request type "GET" and "POST ", if
request type "GET" can be omitted]Example：gc.monitor("lubm");
```

Save the database: checkpoint

```
std::string checkpoint(std::string db_name, std::string request_type);Function：
If you change the database, save the databas Parameter Definition：[database
name], [request type "GET" and "POST ", if request type "GET" can be
omitted]Example：gc.checkpoint("lubm");
```

Show the database: show

```
std::string show(std::string request_type);Function：Displays all created
databases Parameter Definition：[Request types "GET" and "POST" can be omitted
if the request type is "GET"] Example：gc.show();
```

The kernel version information is displayed: getCoreVersion

```
std::string getCoreVersion(std::string request_type);Function：Get kernel
version information Parameter Definition：[Request types "GET" and "POST" can be
omitted if the request type is "GET"]
Example：gc.getCoreVersion();
```

The API version information is displaye: getAPIVersion

```
std::string getAPIVersion(std::string request_type);
Function：Get the API version information
Parameter Definition：[Request types "GET" and "POST" can be omitted if the
request type is "GET"]
Example：gc.getAPIVersion();
```

Query the database and save the file: fquery

```
void fquery(std::string db_name, std::string format, std::string sparql,
std::string filename, std::string request_type);
Function: Query the database and save the results to a file
Parameter Definition: [database name], [query result type JSON, HTML or text],
[SPARQL statement], [file name], [request type "GET" and "POST ", if the request
type is "GET", it can be omitted]
Example: gc.fquery("lubm", "json", sparql, "ans.txt");
```

Exporting the Database

```
std::string exportDB(std::string db_name, std::string dir_path, std::string
request_type);
Function: Export the database to a folder
Parameter Definition: [database name], [path to database export folder],
[request type "GET" and "POST ", if the request type is "GET", can be omitted]
Example: gc.exportDB("lubm", "/root/gStore/");
```

5.5 Java HTTP API

To use the Java API, please refer to the gStore/API/HTTP/Java/SRC/JGSC/GstoreConnector. Java. Specific use is as follows:

Construct the initialization function

```
public class GstoreConnector(String serverIP, int serverPort, String httpType,
String username, String password);
Function: Initialize
Parameter Definition: [Server IP], [HTTP port on the server], [HTTP service
type], [Username], [password]
Example: GstoreConnector gc = new GstoreConnector("127.0.0.1", 9000, "ghttp",
"root", "123456");
```

Building a database: build

```
public String build(String db_name, String rdf_file_path, String request_type);
Function: Create a new database from an RDF file
Parameter Definition: [database name], [.nt file path], [request type "GET" and
"POST ", if request type "GET" can be omitted]
Example: gc.build("lubm", "data/lubm/lubm.nt");
```

Loading a database: load

```
public String load(String db_name, String request_type);
Function: Load the database you created
Parameter Definition: [database name], [request type "GET" and "POST ", if
request type "GET" can be omitted]
Example: gc.load("lubm");
```

Stopping database loading: unload

```
public String unload(String db_name, String request_type);
Function：Stopping database loading
Parameter Definition：[database name], [request type "GET" and "POST ", if
request type "GET" can be omitted]
Example：gc.unload("lubm");
```

User management：user

```
public String user(String type, String username2, String addition, String
request_type);
Function：The root user can add, delete, or modify the user's permission only.
1.Add or delete users：
Parameter Definition：["add_user" adds user, "delete_user" deletes user],[user
name],[password],[request type "GET" and "POST ", if request type "GET" can be
omitted]
Example：gc.user("add_user", "user1", "111111");
2.Modify user's privilege：
Parameter Definition：["Add_query" adds query permission, "delete_query" deletes
query permission, "add_load" adds load permission, "delete_load" deletes load
permission, "add_unload" adds no load permission, "delete_unload" deletes no
load permission,  "Add_update" adds update permission, "delete_update" deletes
update permission, "add_backup" adds backup permission, "delete_bakup" deletes
backup permission, "add_restore" adds restore permission,  "Delete_restore"
deletes restore permission,"add_export" adds export permission," delete_export"
deletes export permission],[user name],[database name],[Request type "GET" and
"POST ", if the request type is "GET", it can be omitted.]
Example：gc.user("add_query", "user1", "lubm");
```

Display user：showUser

```
public String showUser(String request_type);
Function：Display all users
Parameter Definition：[Request types "GET" and "POST" can be omitted if the
request type is "GET"]
Example：gc.showUser();
```

Database query: query

```
public String query(String db_name, String format, String sparql, String
request_type);
Function：query databse
Parameter Definition：[database name], [query result type JSON, HTML or text],
[SPARQL statement], [Request type "GET" and "POST ", if the request type is
"GET", it can be omitted]
Example：
String res = gc.query("lubm", "json", sparql);
System.out.println(res); //output result
```

Database deletion： drop

```
public String drop(String db_name, boolean is_backup, String request_type);
Function：Delete the database directly or delete the database while leaving a
backup.
Parameter Definition：[database name], [false not backup, true backup], [request
type "GET" and "POST ", if request type "GET" can be omitted]
Example：gc.drop("lubm", false);  //Delete the database without leaving a backup
```

Monitoring database: monitor

```
public String monitor(String db_name, String request_type);
Function：Displays information for a specific database
Parameter Definition：[database name], [request type "GET" and "POST ", if
request type "GET" can be omitted]Example：gc.monitor("lubm");
```

Save the database: checkpoint

```
public String checkpoint(String db_name, String request_type);
Function：If you change the database, save the database
Parameter Definition：[database name], [request type "GET" and "POST ", if
request type "GET" can be omitted]Example：gc.checkpoint("lubm");
```

Show database：show

```
public String show(String request_type);Function：Displays all created
databasesParameter Definition：[Request types "GET" and "POST" can be omitted if
the request type is "GET"]Example：gc.show();
```

The kernel version information is displayed：getCoreVersion

```
public String getCoreVersion(String request_type);Function：Get kernel version
information Parameter Definition：[Request types "GET" and "POST" can be omitted
if the request type is "GET"]Example：gc.getCoreVersion();
```

Display API version：getAPIVersion

```
public String getAPIVersion(String request_type);Function：Get API version
Parameter：[Request types "GET" and "POST" can be omitted if the request type is
"GET"]Example：gc.getAPIVersion();
```

Query the database and save the file: fquery

```
public void fquery(String db_name, String format, String sparql, String
filename, String request_type);Function：Query the database and save the result
to a file Parameter definition：[database name], [query result type JSON, HTML
or text], [SPARQL statement], [file name], [Request types "GET" and "POST" can
be omitted if the request type is "GET"]Example：gc.fquery("lubm", "json",
sparql, "ans.txt");
```

Export database

```
public String exportDB(String db_name, String dir_path, String
request_type);Function: Export database to parameter definition under file
folder: [database name], [database export folder path], [Request types "GET" and
"POST" can be omitted if the request type is "GET"】Example: gc.exportDB("lubm",
"/root/gStore/");
```

5.6 Python HTTP API

To use the Python API, please refer to the gStore/API/HTTP/Python/src/GstoreConnector. Py. Specific use is as follows:

Construct the initialization function

```
def __init__(self, ip, port, username, password, http_type='ghttp'):
Function: Initialize
Parameter Definition: [Server IP], [HTTP port on the server], [Username],
[password],[HTTP service type]
Example: gc = GstoreConnector.GstoreConnector("127.0.0.1", 9000, "ghttp",
"root",
"123456")
```

Build database: build

```
def build(self, db_name, rdf_file_path, request_type):
Function: Create a new database from an RDF file
Parameter definition: [Database name], [.nt文件路径], [Request types "GET" and
"POST" can be omitted if the request type is "GET"】
Example: res = gc.build("lubm", "data/lubm/lubm.nt")
```

Load database: load

```
def load(self, db_name, request_type):
Function: load the database you have created
Parameter definition: [database name], [request type "GET" and "POST ", if
request type "GET" can be omitted]
Example: res = gc.load("lubm")
```

Unload database: unload

```
def unload(self, db_name, request_type):
Function: Unload database
Parameter definition: [database name], [request type "GET" and "POST ", if
request type "GET" can be omitted]
Example: res = gc.unload("lubm")
```

User management: user

```
def user(self, type, username2, addition, request_type):
Function: The root user can add, delete, or modify the user's permission only.
1.Add or delete users:
Parameter definition: ["add_user" adds a user, "delete_user" deletes a user],
[username],[password],[Request types "GET" and "POST" can be omitted if the
Request type is "GET"]
Example: res = gc.user("add_user", "user1", "111111")
2.Modify user's privilege:
Parameter definition: ["Add_query" adds query permission, "delete_query" deletes
query permission, "add_load" adds load permission, "delete_load" deletes load
permission, "add_unload" adds no load permission, "delete_unload" deletes no
load permission,  "Add_update" adds update permission, "delete_update" deletes
update permission, "add_backup" adds backup permission, "delete_bakup" deletes
backup permission, "add_restore" adds restore permission,  "Delete_restore"
deletes restore permission,"add_export" adds export permission," delete_export"
deletes export permission],  [user name],[database name],[Request types "GET"
and "POST" can be omitted if the Request type is "GET"]
Example: res = gc.user("add_query", "user1", "lubm")
```

Display users: showUser

```
def showUser(self, request_type):
Function: Display all users
Parameter definition: [Request types "GET" and "POST" can be omitted if the
request type is "GET"]
Example: res = gc.showUser()
```

Query Database: query

```
def query(self, db_name, format, sparql, request_type):
Function: Query the database
Parameter definition: [Database name], [Query result type JSON, HTML or text],
[SPARQL statement], [Request types "GET" and "POST" can be omitted if the
Request type is "GET"]
Example:
res = gc.query("lubm", "json", sparql)
print(res) //output result
```

Database deletion: drop

```
def drop(self, db_name, is_backup, request_type):
Function: Delete the database directly or delete the database while leaving a
backup
Parameter definition: [database name], [false not backup, true backup],[Request
types "GET" and "POST" can be omitted if the request type is "GET"]
Example: res = gc.drop("lubm", false)  //Delete the database without leaving a
backup
```

Database Monitor： monitor

```
def monitor(self, db_name, request_type):    Function: Displays information for
a specific database Parameter definition: [Database name], [Request types "GET"
and "POST" can be omitted if the request type is "GET"]
Example: res = gc.monitor("lubm")
```

Save database: checkpoint

```
def checkpoint(self, db_name, request_type):Function: If the database is
changed, save the meaning of the database parameters: [database name], [request
type "GET" and "POST ", if request type "GET" can be omitted] Example: res =
gc.checkpoint("lubm")
```

Display database: show

```
def show(self, request_type):Function: Display the meanings of all created
databases: [Request types "GET" and "POST" can be omitted if the request type is
"GET"]
Example: res = gc.show()
```

The kernel version information is displayed: getCoreVersion

```
def getCoreVersion(self, request_type):Function: Get the definition of kernel
version parameter : [Request types "GET" and "POST" can be omitted if the
request type is "GET"]
Example: res = gc.getCoreVersion()
```

Display API version: getAPIVersion

```
def getAPIVersion(self, request_type):Function: Get the API version information
parameter definition: [Request types "GET" and "POST" can be omitted if the
request type is "GET"]
Example: res = gc.getAPIVersion()
```

Query the database and save the file: fquery

```
def fquery(self, db_name, format, sparql, filename, request_type):Function:
Query the database and save the result to a file Parameter definition: [database
name], [query result type JSON, HTML or text], [SPARQL statement], [file name],
[Request types "GET" and "POST" can be omitted if the request type is "GET"]
Example: gc.fquery("lubm", "json", sparql, "ans.txt")
```

Export databse

```
def exportDB(self, db_name, dir_path, request_type): Function: parameter
definition of exprotin database to folders: [database name], [database export
folder path], [Request types "GET" and "POST" can be omitted if the request type
is "GET"]
Example: res = gc.exportDB("lubm", "/root/gStore/")
```

5.7 Node.js HTTP API

Before using the Nodejs API, type `NPM install Request` and `NPM Install request-promise` to add the required modules under the Nodejs folder.

To use Nodejs API, please refer to the `gStore/API/http/Nodejs/GstoreConnector.js`. Specific use is as follows:

Construct the initialization function

```
class GstoreConnector(ip = '', port, httpType = 'ghttp', username = '', password
= '');
Function: Initialize
Parameter Definition：[Server IP], [HTTP port on the server], [HTTP service
type], [Username], [password]
Example：gc = new GstoreConnector("127.0.0.1", 9000, "ghttp", "root", "123456");
```

Build database: build

```
async build(db_name = '', rdf_file_path = '', request_type);
Function：Create a new database from an RDF file
The defintion of parameters are as follows: [database name], [.nt file path],
[Request types "GET" and "POST" can be omitted if the request type is "GET"]
Example：res = gc.build("lubm", "data/lubm/lubm.nt");
```

Load database：load

```
async load(db_name = '', request_type);
Function：Load the database you have created
Parameter definition：[database name], [request type "GET" and "POST ", if
request type "GET" can be omitted]
Example：res = gc.load("lubm");
```

Unload databse：unload

```
async unload(db_name = '', request_type);
Function：Unload databse
Parameter definition：[database name], [request type "GET" and "POST ", if
request type "GET" can be omitted]
Example：res = gc.unload("lubm");
```

User management：user

```
async user(type = '', username2 = '' , addition = '' , request_type);
Function：The root user can add, delete, or modify the user's permission only
1.Add or delete users：
Parameter definition：["add_user" adds a user, "delete_user" deletes a user],
[user name],[password],[Request types "GET" and "POST" can be omitted if the
Request type is "GET"]
Example：res = gc.user("add_user", "user1", "111111");
2.Privilege to modify user：
Parameter definition：["Add_query" adds query permission, "delete_query" deletes
query permission, "add_load" adds load permission, "delete_load" deletes load
permission, "add_unload" adds no load permission, "delete_unload" deletes no
load permission,  "Add_update" adds update permission, "delete_update" deletes
update permission, "add_backup" adds backup permission, "delete_bakup" deletes
backup permission, "add_restore" adds restore permission,  "Delete_restore"
deletes restore permission,"add_export" adds export permission," delete_export"
deletes export permission],  [user name],[database name],[Request types "GET"
and "POST" can be omitted if the Request type is "GET"]]
Example：res = gc.user("add_query", "user1", "lubm");
```

Display user：showUser

```
async showUser(request_type);
Function：Display all users
Parameter definition: [Request types "GET" and "POST" can be omitted if the
request type is "GET"]
Example：res = gc.showUser();
```

Query database：query

```
async query(db_name = '', format = '' , sparql = '' , request_type);
Function：Query database
Parameter definition：[Database name], [query result type JSON, HTML or text],
[SPARQL statement], [Request types "GET" and "POST" can be omitted if the
Request type is "GET"]
Example：
res = gc.query("lubm", "json", sparql);
console.log(JSON.stringify(res,",")); //output result
```

Database deletion：drop

```
async drop(db_name = '', is_backup , request_type);
Function：Delete the database directly or delete the database while leaving a
backup
Parameter definition：[database name]，[false no backup, true backup]，[Request
types "GET" and "POST" can be omitted if the request type is "GET"]
Example：res = gc.drop("lubm", false);  //Delete the database without leaving a
backup
```

Database monitor: monitor

```
async monitor(db_name = '', request_type);     Function：Parameter definition
for displaying information about a specific database：[database name], [request
type "GET" and "POST ", if request type "GET" can be omitted]Example：res =
gc.monitor("lubm");
```

Save database: checkpoint

```
async checkpoint(db_name = '', request_type);Function：If the database is
changed, the parameter definition of saving database：[database name], [request
type "GET" and "POST ", if request type "GET" can be omitted]Example：res =
gc.checkpoint("lubm");
```

Display database: show

```
async show(request_type);Function：Displays all created databases Parameter
definition: [Request types "GET" and "POST" can be omitted if the request type
is "GET"]Example：res = gc.show();
```

Display kernel version information: getCoreVersion

```
async getCoreVersion(request_type);Function: Get kernel version information
Parameter definition: [Request types "GET" and "POST" can be omitted if the
request type is "GET"]Example: res = gc.getCoreVersion();
```

Display API version: getAPIVersion

```
async getAPIVersion(request_type);
Function: Get the API version information
Parameter definition: [Request types "GET" and "POST" can be omitted if the
request type is "GET"]
Example: res = gc.getAPIVersion();
```

Query the database and save the file: fquery

```
async fquery(db_name = '', format = '' , sparql = '' , filename = '' ,
request_type);
Function: Query the database and save the results to a file
Parameter definition: [database name], [query result type JSON, HTML or text],
[SPARQL statement], [file name],  [Request types "GET" and "POST" can be omitted
if the Request type is "GET"]
Example: gc.fquery("lubm", "json", sparql, "ans.txt");
```

Export database

```
async exportDB(db_name = '' , dir_path = '' , request_type);
Function: export database to folders
parameter definition: [database name], [directory where the database is
exported], [Request types "GET" and "POST" can be omitted if the Request type is
"GET"]
Example: res = gc.exportDB("lubm", "/root/gStore/");
```

5.8 PHP HTTP API

To use the Php API, please refer to the gStore/API/HTTP/Php/SRC/GstoreConnector. Php. Specific use is as follows:

Construct the initialization function

```
class GstoreConnector($ip, $port, $httpType, $username, $password)
Function: Initialize
Parameter Definition: [Server IP], [HTTP port on the server], [HTTP service
type], [Username], [password]
Example: $gc = new GstoreConnector("127.0.0.1", 9000, "ghttp", "root",
"123456");
```

Build databse：build

```
function build($db_name, $rdf_file_path, $request_type)
Function：Create a new database from an RDF file
Parameter definition：[database name], [.nt file path],[Request types "GET" and
"POST" can be omitted if the request type is "GET"]
Example：
$res = $gc->build("lubm", "data/lubm/lubm.nt");
echo $res . PHP_EOL;
```

Load database：load

```
function load($db_name, $request_type)
Function：Load the database you have created
Parameter definition：[database name], [request type "GET" and "POST ", if
request type "GET" can be omitted]
Example：
$ret = $gc->load("test");
echo $ret . PHP_EOL;
```

Unload databse：unload

```
function unload($db_name, $request_type)
Function：Unload databse
Parameter definition：[database name]，[Request types "GET" and "POST" can be
omitted if the request type is "GET"]
Example：
$ret = $gc->unload("test");
echo $ret . PHP_EOL;
```

User management：user

```
function user($type, $username2, $addition, $request_type)
Function：Only the root user can add, delete, or modify the user's permission
1.Add or delete users：
Parameter definition：["add_user" adds a user, "delete_user" deletes a user],
[user name],[password],[Request types "GET" and "POST" can be omitted if the
Request type is "GET"]
Example：
$res = $gc->user("add_user", "user1", "111111");
echo $res . PHP_EOL;
2.Privilege to modify user：
参数含义：["Add_query" adds query permission, "delete_query" deletes query
permission, "add_load" adds load permission, "delete_load" deletes load
permission, "add_unload" adds no load permission, "delete_unload" deletes no
load permission,  "Add_update" adds update permission, "delete_update" deletes
update permission, "add_backup" adds backup permission, "delete_bakup" deletes
backup permission, "add_restore" adds restore permission,  "Delete_restore"
deletes restore permission,"add_export" adds export permission," delete_export"
deletes export permission],  [user name],[database name],[Request types "GET"
and "POST" can be omitted if the Request type is "GET"]]
Example：
$res = $gc->user("add_user", "user1", "lubm");
echo $res . PHP_EOL;
```

Display user：showUser

```
function showUser($request_type)
Function：Display all users
Parameter definition：[Request types "GET" and "POST" can be omitted if the
request type is "GET"]
Example：
$res = $gc->showUser();
echo $res. PHP_EOL;
```

Query database：query

```
function query($db_name, $format, $sparql, $request_type)
Parameter definition：[database name], [query result type JSON, HTML or text],
[SPARQL statement],[Request types "GET" and "POST" can be omitted if the request
type is "GET"]
Example：
$res = $gc->query("lubm", "json", $sparql);
echo $res. PHP_EOL; //output result
```

Database deletion：drop

```
function drop($db_name, $is_backup, $request_type)
Function：Delete the database directly or delete the database while leaving a
backup
Parameter definition：[database name], [false not backup, true backup],[Request
types "GET" and "POST" can be omitted if the request type is "GET"]
Example：
$res = $gc->drop("lubm", false); //Delete the database without leaving a backup
echo $res. PHP_EOL;
```

Database monitor: monitor

```
function monitor($db_name, $request_type)Function：Displays information for a
specific database
Parameter definition：[database name], [request type "GET" and "POST ", if
request type "GET" can be omitted]
Example：$res = $gc->monitor("lubm");echo $res. PHP_EOL;
```

Save database: checkpoint

```
function checkpoint($db_name, $request_type)Function：the Parameter definition
of save database if it has been changed：[database name], [request type "GET"
and "POST ", if request type "GET" can be omitted]Example：$res = $gc-
>checkpoint("lubm");echo $res. PHP_EOL;
```

Display database: show

```
function show($request_type)Function：Displays all created databases
Parameter definition：[Request types "GET" and "POST" can be omitted if the
request type is "GET"]
Example：$res = $gc->show();echo $res. PHP_EOL;
```

Display kernel version information: getCoreVersion

```
function getCoreVersion($request_type)Function：get kernel version information
Parameter definition: [Request types "GET" and "POST" can be omitted if the
request type is "GET"]
Example：$res = $gc->getCoreVersion();echo $res. PHP_EOL;
```

Display API version: getAPIVersion

```
function getAPIVersion($request_type)
Function：Get API version
Parameter definition: [Request types "GET" and "POST" can be omitted if the
request type is "GET"]
Example：
$res = $gc->getAPIVersion();
echo $res. PHP_EOL;
```

Query the database and save the file：fquery

```
function fquery($db_name, $format, $sparql, $filename, $request_type)
Function：Query the database and save the results to a file
Parameter definition：[database name], [query result type JSON, HTML or text],
[SPARQL statement], [file name],[Request types "GET" and "POST" can be omitted
if the request type is "GET"]
Example：$gc->fquery("lubm", "json", $sparql, "ans.txt");
```

Export database

```
function exportDB($db_name, $dir_path, $request_type)
Function：Export the database to a folder
Parameter definition：[database name], [database export folder path]，[Request
types "GET" and "POST" can be omitted if the request type is "GET"]
Example：$res = $gc->exportDB("lubm", "/root/gStore/");
```

6. SPARQL query syntax

6.1 Graph Patterns

This document mainly refer to the SPARQL 1.1 Standard document, but also increased the gStore own customized content, if you want to learn more about gStore SPARQL statement of support, Please read our documentation carefully!

Unless otherwise specified, this document will continue to use the following RDF data instances as objects for queries:

```
<YifeiLiu> <name> "Yifei Liu" .
<YifeiLiu> <name> "Crystal Liu" .
<YifeiLiu> <gender> "female" .
<YifeiLiu> <constellation> "Virgo" .
<YifeiLiu> <occupation> "actor" .

<ZhiyingLin> <name> "Zhiying Lin" .
<ZhiyingLin> <gender> "male" .
<ZhiyingLin> <occupation> "actor" .
<ZhiyingLin> <occupation> "director" .

<JunHu> <name> "Jun Hu" .
<JunHu> <gender> "male" .
<JunHu> <constellation> "pisces" .
<JunHu> <occupation> "actor" .
<JunHu> <occupation> "voice actors" .
<JunHu> <occupation> "producer" .
<JunHu> <occupation> "director" .

<TianlongBabu> <featured> <ZhiyingLin> .
<TianlongBabu> <featured> <YifeiLiu> .
<TianlongBabu> <featured> <JunHu> .
<TianlongBabu> <type> <ActionMovie> .
<TianlongBabu> <type> <CostumeFilms> .
<TianlongBabu> <type> <RomanticMovie> .
<TianlongBabu> <score> "8.3"^^<http://www.w3.org/2001/XMLSchema#float> .
<TianlongBabu> <releaseTime> "2003-12-
11T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> .

<TheLoveWinner> <featured> <ZhiyingLin> .
<TheLoveWinner> <featured> <YifeiLiu> .
<TheLoveWinner> <type> <RomanticMovie> .
<TheLoveWinner> <type> <FeatureFilm> .
<TheLoveWinner> <score> "6.1"^^<http://www.w3.org/2001/XMLSchema#float> .
<TheLoveWinner> <releaseTime> "2004-11-
30T00:00:00"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
```

Since there is no official Chinese translation of the SPARQL 1.1 standard document, the English version of the term will be indicated when it first appears in the following paragraphs.

By standard, **keywords in SPARQL queries are case insensitive**

6.1.1 The simplest graph mode

Let's start with the simplest query:

```
SELECT ?movie
WHERE
{
    ?movie <featured> <YifeiLiu> .
}
```

The query consists of two parts: the **SELECT statement** specifies the variables that need to output the query results, and the **WHERE statement** provides the graph pattern used to match the data graph. In the above query, the graph pattern consists of a single **triplet** `?movie <featured> <YifeiLiu>` `?movie` is **variable** , and `<featured>` as predicate and `<YifeiLiu>` as object are **IRI** (International Resource Identifier). This query will return all movies and TV works starring Yifei Liui. The results run on the sample data are as follows:

?movie
<TianlongBabu>
<TheLoveWinner>

The subject, predicate and object of a triple can all be IRI. Objects can also be **RDF literals** . The following query will give all the people in the sample data whose profession is director:

```
SELECT ?person
WHERE
{
    ?person <occupation> "director" .
}
```

Where 'director' is an RDF literal

Result are as follows:

?person
<JunHu>
<ZhiyingLin>

Under the current version of gStore, RDF literals with data types are queried with suffixes corresponding to those in the data file. For example, the following query will give a douban rating of 8.3:

```
SELECT ?movie
WHERE
{
    ?movie <score> "8.3"^^<http://www.w3.org/2001/XMLSchema#float> .
}
```

Results are as follows:

?movie
<TianlongBabu>

Other common data types include `<http://www.w3.org/2001/XMLSchema#integer>` (integer), `<http://www.w3.org/2001/XMLSchema#decimal>` (point type), `xsd:double` (A double-precision floating point type), `<http://www.w3.org/2001/XMLSchema#string>` (String type), `<http://www.w3.org/2001/XMLSchema#boolean>` (Boolean), `<http://www.w3.org/2001/XMLSchema#dateTime>` (Date/time).Other data types may also appear in data files, simply using the form `^^< data type suffix >` in the query.

6.1.2 Basic Graph Pattern

Base graph pattern is a collection of triples; The two queries in the previous section both have only the outermost braces and therefore belong to the **basic graph mode**; Enclosing the outermost braces is a single base graph pattern **Group Graph Pattern。**

The basic graph pattern in the two queries in the previous section consists of a single triple. The following query uses a basic graph pattern consisting of multiple triples to give all male leads of Tianlongba in the sample data:

```
SELECT ?person
WHERE
{
    <TianlongBabu> <featured> ?person .
    ?person <gender> "male" .
}
```

Result are as follows:

?person
<JunHu>
<ZhiyingLin>

6.1.3 Group Graph Pattern

Group graph patterns are separated by paired braces. A group graph pattern can be composed of a single base graph pattern, as described in the previous section, or multiple subgroups of graph patterns nested with the following operations: **OPTIONAL**, **UNION**, and **MINUS**. **FILTER** filters the results within the range of a group graph pattern

OPTIONAL

The keyword OPTIONAL uses the following syntax:

```
pattern1 OPTIONAL { pattern2 }
```

The result of the query must match `pattern1` and selectively match `pattern2`. `Pattern2` is known as the OPTIONAL graph pattern. If there is a match for `pattern2`, add it to the match for `Pattern1`; Otherwise, the match for `pattern1` is still printed. For this reason, OPTIONAL is often used when some data is missing.

The following query gives the gender and constellation information of the person in the sample data. Among them, as long as there is gender information of the character will be returned, regardless of whether there is the constellation information of the character; If both exist, additional returns are returned

```
SELECT ?person ?gender ?horoscope
WHERE
{
    ?person <gender> ?gender .
    OPTIONAL
    {
        ?person <constellation> ?horoscope .
    }
}
```

Results are as follows:

?person	?gender	?horoscope
<YifeiLiu>	"female"	"Virgo"
<ZhiyingLin>	"male"	
<JunHu>	"male"	"pisces"

UNION

The keyword UNION is syntactically similar to OPTIONAL. In a graph pattern joined by UNION, as long as there is one that matches a piece of data, that data matches the whole joined by UNION. Therefore, UNION can be understood as finding the set of matching results of each graph pattern it joins (actually using multiple set semantics because it allows repeating results).

The following query gives the sample data for films and television works whose category is costume film or drama film:

```
SELECT ?movie
WHERE
{
    {?movie <type> <CostumeFilms> .}
    UNION
    {?movie <type> <FeatureFilm> .}
}
```

Results are as follows:

?movie
<Tianlong Babu>
<The Love Winner>

MINUS

The usage syntax of the keywords MINUS is similar to OPTIONAL and UNION. MINUS The matching of the left and right graph patterns will be calculated, and the part that can match the right graph pattern will be removed from the matching result of the left graph pattern as the final result. Therefore, MINUS can be understood as the difference of the matching result set of the two graph patterns connected to it (the left is the subtracted set, multiple set semantics).

The following query will give the sample data of the characters who starred in The Dragon Eight but did not star in the love winner:

```
SELECT ?person
WHERE
{
    <Tianlong Babu> <featured> ?person .
    MINUS
    {<The Love Winner> <featured> ?person .}
}
```

Results are as follows:

?person
<JunHu>

FILTER

The keyword FILTER is followed by a constraint condition, and the results that do not meet this condition in the current pattern group will be filtered out and not returned. FILTER conditions can use equations, inequalities, and various built-in functions.

The following query will give the film and television works with douban score higher than 8 points in the sample data:

```
SELECT ?movie
WHERE
{
    ?movie <score> ?score .
    FILTER (?score > "8"^^<http://www.w3.org/2001/XMLSchema#float>)
}
```

Results are as follows：

?movie
<TianlongBabu>

No matter where a FILTER is placed in a group graph pattern, as long as it remains in the same nesting layer, its semantics remain unchanged and the scope of the constraint remains the current group graph pattern. For example, the following query is equivalent to the previous one:

```
SELECT ?movie
WHERE
{
    FILTER (?score > "8"^^<http://www.w3.org/2001/XMLSchema#float>)
    ?movie <score> ?score .
}
```

One of the built-in functions commonly used for FILTER conditions is the regular expression **REGEX**. The following query gives the liu surname in the sample data:

```
SELECT ?person
WHERE
{
    ?person <name> ?name .
    FILTER REGEX(?name, "Yi.*")
}
```

Results are as follows:

?person
<YifeiLiu>

6.1.4 Built-in Function

(1) ISIRI

Checks if the data is an IRI.

Example:

The following query checks if the lead actors of "Demi-Gods and Semi-Devils" in the sample data are IRIs:

```
SELECT ?name
WHERE
{
    <Demi-Gods and Semi-Devils> <LeadActor> ?name .
    FILTER ISIRI(?name)
}
```

The results are as follows:

?name
<Liu Yifei>
<Jimmy Lin>
<Hun Jun>

(2) ISLITERAL

Checks if the data is a literal.

Example:

The following query checks if the names of Liu Yifei in the sample data are string literals:

```
SELECT ?name
WHERE
{
    <Liu Yifei> <Name> ?name .
    FILTER ISLITERAL(?name)
}
```

The results are as follows:

?name
"Liu Yifei"
"Crystal Liu"

(3) ISNUMERIC

Checks if the data is of numeric type.

Example:

The following query retrieves all data from the sample where the object is a numeric type:

```
SELECT ?p, ?x, ?name
WHERE
{
    ?p ?x ?name .
    FILTER ISNUMERIC(?name)
}
```

结果如下:

?p	?x	?name
<Love Winner>	<DoubanScore>	"6.1"^^http://www.w3.org/2001/XMLSchema#float
<Demi-Gods and Semi-Devils>	<DoubanScore>	"8.3"^^http://www.w3.org/2001/XMLSchema#float

(4) CONCAT

Concat multiple characters

```
CONCAT(val_1, val_2,...val_n)
```

Parameter

`val_i`: string-typed string value

Example:

Connect the name, gender and profession of the people found with each other:

```
SELECT (CONCAT(?name, ",", str(?gender), ",", ?profession) as ?info) WHERE
{
    ?s <name> ?name.
    ?s <gender> ?gender.
    ?s <occupation> ?profession.
}
```

The final result output is as follows (for ease of reading, the outer double quotation marks and the inner double quotation escape in the string are omitted):

```
{
    "bindings": [
        {
            "info": {"type": "literal","value": "YifeiLiu,female,actor"}
        },
        {
            "info": {"type": "literal","value": "Crystal Liu,female,actor"}
        },
        {
            "info": {"type": "literal","value": "ZhiyingLin,male,actor"}
        },
        {
            "info": {"type": "literal","value": "ZhiyingLin,male,director"}
        },
        {
            "info": {"type": "literal","value": "JunHu,male,actor"}
        },
        {
            "info": {"type": "literal","value": "JunHu,male,director"}
        },
        {
            "info": {"type": "literal","value": "JunHu,male,voice actors"}
        },
        {
            "info": {"type": "literal","value": "JunHu,male,producer"}
        }
    ]
}
```

6.2 Assignment

The following keywords belong to assignment functions and can be used to define variables in the query body or provide inline data.

(1) BIND: assigning to a variable

```
BIND(value, name)
```

Parameter

`value`: a string value
`name`: name of a variable that has not yet appeared in the query

Sample 1

Query the occupation of YifeiLiu, JunHu and categorize the tags in the results returned:

```
SELECT ?a ?x WHERE
{
    {
        BIND("YifeiLiu" as ?info).
        <YifeiLiu> <occupation> ?profession.
    }
    UNION
    {
        BIND("JunHu" as ?info).
        <JunHu> <accupation> ?profession.
    }
}
```

The result is as follows: (For the convenience of reading, the escaping of the outermost double quotes and the inner double quotes of the string is omitted)

```
{
    "bindings": [
        {
            "info": {"type": "literal", "value": "YifeiLiu"},
            "profession": {"type": "literal", "value": "actor"}
        },
        {
            "info": {"type": "literal", "value": "JunHu"},
            "profession": {"type": "literal", "value": "actor"}
        },
        {
            "info": {"type": "literal", "value": "JunHu"},
            "profession": {"type": "literal", "value": "voice actors"}
        },
        {
            "info": {"type": "literal", "value": "JunHu"},
            "profession": {"type": "literal", "value": "producer"}
        },
        {
            "info": {"type": "literal", "value": "JunHu"},
            "profession": {"type": "literal", "value": "director"}
        }
    ]
}
```

Sample 2

Binding entity(IRI) query the occupation of YifeiLiu, ZhiyingLin and categorize the tags in the results returned:

```
SELECT ?info ?profession WHERE
{
    {
        BIND("YifeiLiu" as ?info).
        <YifeiLiu> <occupation> ?profession.
    }
    UNION
    {
        BIND("ZhiyingLin" as ?info).
        <ZhiyingLin> <accupation> ?profession.
    }
}
```

The result is as follows: (For the convenience of reading, the escaping of the outermost double quotes and the inner double quotes of the string is omitted)

```
{
    "bindings": [
        {
            "info": {"type": "literal", "value": "YifeiLiu"},
            "profession": {"type": "literal", "value": "actor"}
        },
        {
            "info": {"type": "literal", "value": "ZhiyingLin"},
            "profession": {"type": "literal", "value": "actor"}
        },
        {
            "info": {"type": "literal", "value": "ZhiyingLin"},
            "profession": {"type": "literal", "value": "voice actors"}
        }
    ]
}
```

6.3 Aggregates

Aggregate functions are used in SELECT statements with the following syntax:

```
SELECT (AGGREGATE_NAME(?x) AS ?y)
WHERE
{
    ...
}
```

Where `AGGREGATE_NAME` is the name of the aggregation function, variable `?X` is the aggregate function on the object, variable `?Y` is the column name of the aggregate function value in the final result.

The aggregate function acts on each group of outcomes. All results are returned as a set by default. The aggregation functions supported by gStore are as follows

COUNT

Aggregate function for counting.

The following query gives the number of actors in the sample data:

```
SELECT (COUNT(?person) AS ?count_person)
WHERE
{
    ?person <occupation> "actor" .
}
```

Results are as follows:

?count_person
"3"^^<http://www.w3.org/2001/XMLSchema#integer>

SUM

Aggregate function for summation.

The following query will give the sum of douban ratings for all movies in the sample data:

```
SELECT (SUM(?score) AS ?sum_score)
WHERE
{
    ?movie <score> ?score .
}
```

Results are as follows:

?sum_score
"14.400000"^^http://www.w3.org/2001/XMLSchema#float

AVG

An aggregate function for averaging.

The following query gives the average Douban score for all movies in the sample data:

```
SELECT (AVG(?score) AS ?avg_score)
WHERE
{
    ?movie <score> ?score .
}
```

Results are as follows:

?avg_score
"7.200000"^^http://www.w3.org/2001/XMLSchema#float

MIN

An aggregate function for minimizing.

The following query will give the lowest Douban score for all movies in the sample data:

```
SELECT (MIN(?score) AS ?min_score)
WHERE
{
    ?movie <score> ?score .
}
```

Results are as follows:

?min_score
"6.1"^^http://www.w3.org/2001/XMLSchema#float

MAX

Aggregate function used to find the maximum value.

The following query will give the highest Douban ratings for all movies in the sample data:

```
SELECT (MAX(?score) AS ?max_score)
WHERE
{
    ?movie <score> ?score .
}
```

Results are as follows:

?max_score
"8.3"^^http://www.w3.org/2001/XMLSchema#float

GROUP BY

If you want to GROUP the results BY the value of a variable, you can use the keyword GROUP BY. For example, the following query gives all occupations and their corresponding numbers in the sample data:

```
SELECT ?occupation (COUNT(?person) AS ?count_person)
WHERE
{
    ?person <occupation> ?occupation .
}
GROUP BY ?occupation
```

Results are as follows:

?occupation	?count_person
"actor"	"3"^^<http://www.w3.org/2001/XMLSchema#integer>
"director"	"2"^^<http://www.w3.org/2001/XMLSchema#integer>
"voice actors"	"1"^^<http://www.w3.org/2001/XMLSchema#integer>
"producer"	"1"^^<http://www.w3.org/2001/XMLSchema#integer>

SAMPLE

If you want to randomly return one value from a set that contains multiple values, you can use the SAMPLE keyword. For example, the following query randomly returns one movie name from the films that have a Douban score:

```
SELECT (SAMPLE(?movie) AS ?sample_movie)
WHERE
{
?movie <score> ?score .
}
```

```
------------------
|  ?sample_movie |
------------------
| <Tanlongbabu>     |
------------------
```

6.4 Solution Sequences and Modifiers

The following keywords are result sequence modifiers that post process the query results to form the final returned results.

DISTINCT: Removes duplicate results

A query with a SELECT statement without the keyword DISTINCT will retain duplicate results in the final result. For example, the following query gives all the occupations in the sample data

```
SELECT ?occupation
WHERE
{
    ?person <occupation> ?occupation .
}
```

Results are as follows:

?occupation
"actor"
"actor"
"actor"
"director"
"director"
"producer"
"voice actors"

If you want to see DISTINCT job categories, you can add the keyword DISTINCT in the SELECT statement:

```
SELECT DISTINCT ?occupation
WHERE
{
    ?person <occupation> ?occupation .
}
```

Results are as follows:

?occupation
"actor"
"director"
"producer"
"voice actors"

DISTINCT can also be used in the aggregate function COUNT. The following query gives the number of occupations in the sample data:

```
SELECT (COUNT(DISTINCT ?occupation) AS ?count_occupation)
WHERE
{
    ?person <occupation> ?occupation .
}
```

Results are as follows:

?count_occupation
"4"^^<http://www.w3.org/2001/XMLSchema#integer>

ORDER BY: 排序

Query results are unordered by default. If you want to sort the results based on the values of some variables, you can add an ORDER BY statement after the WHERE statement. For example, the following query will sort the film and television works in the sample data according to douban score. If the order is not specified, it will be in ascending order by default

```
SELECT ?movie ?score
WHERE
{
    ?movie <score> ?score
}
ORDER BY ?score
```

Results are as follows:

?movie	?score
<The LoveWinner>	"6.1"^^<http://www.w3.org/2001/XMLSchema#float>
<TianlongBabu>	"8.3"^^<http://www.w3.org/2001/XMLSchema#float>

If you want to sort in descending order, you need to modify the variable name with the keyword DESC:

```
SELECT ?movie ?score
WHERE
{
    ?movie <score> ?score
}
ORDER BY DESC(?score)
```

Results are as follows:

?movie	?score
<TianlongBabu>	"8.3"^^<http://www.w3.org/2001/XMLSchema#float>
<TheLoveWinner>	"6.1"^^<http://www.w3.org/2001/XMLSchema#float>

The ORDER BY statement can contain multiple space-separated variables, each of which can be decorated with DESC. gStore does not currently support the use of four-operation expressions and built-in functions in ORDER BY statements.

OFFSET: skips a certain number of results

The OFFSET statement follows the WHERE statement and has the following syntax:

```
OFFSET nonnegative_integer
```

`nonnegative_INTEGER` must be a non-negative integer, indicating the number of results to be skipped. `OFFSET 0` is syntactic but has no effect on the result. Because the query results are unordered by default, SPARQL semantics do not guarantee that the skipped results meet any deterministic conditions. Therefore, the OFFSET statement is typically used in conjunction with the ORDER BY statement。

The following query sorts the film and television works in the sample data by douban score from lowest to highest, and skips the film and television works with the lowest score:

```
SELECT ?movie ?score
WHERE
{
    ?movie <score> ?score .
}
ORDER BY ?score
OFFSET 1
```

Results are as follows:

?movie	?score
<TianlongBabu>	"8.3"^^<http://www.w3.org/2001/XMLSchema#float>

LIMIT: Limit the number of results

The syntax of the LIMIT statement is similar to that of the OFFSET statement:

```
LIMIT nonnegative_integer
```

`nonnegative_INTEGER` must be a non-negative integer, indicating the maximum number of results allowed. Similar to OFFSET, the LIMIT statement is typically used in conjunction with the ORDER BY statement because the query result defaults to unordered.

The following query gives the film and television works with the highest douban score in the sample data:

```
SELECT ?movie ?scoreWHERE{  ?movie <score> ?score .}ORDER BY DESC(?score)LIMIT 1
```

Results are as follows:

?movie	?score
<TianlongBabu>	"8.3"^^<http://www.w3.org/2001/XMLSchema#float>

6.5 Update graph

By **INSERT DATA**, **DELETE DATA**, and **DELETE WHERE** queries, we can INSERT or DELETE triples from the database.

INSERT DATA

INSERT DATA is used to INSERT triples into a database. The syntax is similar to that of a SELECT query, except that there are no variables in the triples that make up the group graph pattern.

The following query inserts the relevant information of the film and TELEVISION works Chinese Paladin into the sample data:

```
INSERT DATA
{
    <Paladin> <featured> <GeHu> .
    <Paladin> <featured> <YifeiLiu> .
    <Paladin> <type> <ActionMovie> .
    <Paladin> <type> <CostumeFilms> .
    <Paladin> <type> <RomanticMovie> .
    <Paladin> <score> "8.9"^^<http://www.w3.org/2001/XMLSchema#float> .
}
```

The query that appears in the "Graph pattern - The simplest Graph pattern" section

```
SELECT ?movie
WHERE
{
    ?movie <featured> <YifeiLiu> .
}
```

After inserting the above data, the result becomes:

?movie
<TianlongBabu>
<TheLoveWinner>
<Paladin>

INSERT WHRER

The INSERT WHERE clause is used to insert triples into a database that meet specific conditions. Unlike INSERT DATA, the WHERE clause in INSERT WHERE is identical to the WHERE clause in a SELECT query, meaning that the triples can include variables. For example, the following query inserts country information for martial arts films in the sample data:

```
INSERT
{
    ?movie <country> "China" .
}
WHERE
{
    ?movie <type> <ActionMovie> .
}
```

DELETE DATA

DELETE DATA is used to DELETE triples from a database. The usage is exactly similar to INSERT DATA.

DELETE WHERE

DELETE DATA is used to DELETE eligible triples from the database; In contrast to DELETE DATA, its WHERE statement is exactly the same as the WHERE statement of a SELECT query, meaning that variables are allowed in triples. For example, the following query removes all information about swordsman films from the sample data:

```
DELETE WHERE{   ?movie <type> <ActionMovie> .   ?movie ?y ?z .}
```

Run the query that appeared in the "Graph patterns - The simplest Graph patterns" section again:

```
SELECT ?movie
WHERE
{
    ?movie <featured> <YifeiLiu> .
}
```

Result change to:

?movie
<TheLoveWinner>

6.6 Advanced functions

In **kernel version V0.9.1**, gStore has added a number of queries related to the path and centrality of nodes in the data graph, including loop query, shortest path query, K-hop reachable query and Personalized PageRank query.

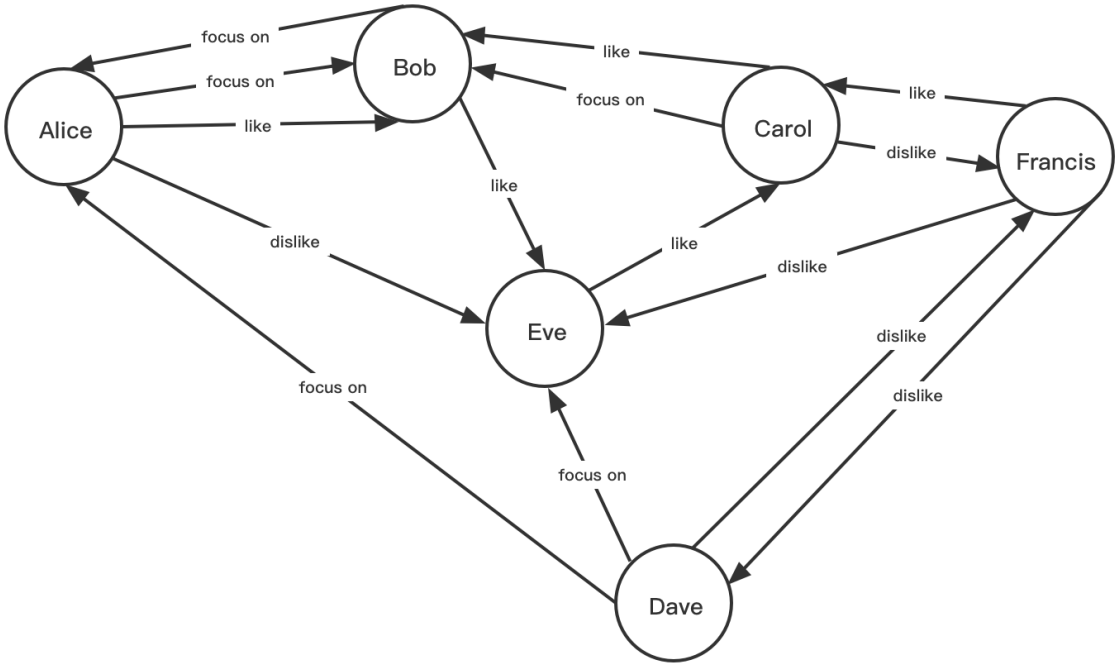
When using advanced functions, you need to load the CSR resources. When starting the HTTP API service, you need to add the parameter `-c 1`. Please see [Quick Start] - [Common API] - [HTTP API service] for details.

6.6.1 Sample data

To better demonstrate the advanced functionality, use the following social relationship data as sample data:

```
<Alice> <focus> <Bob> .
<Alice> <like> <Bob> .
<Alice> <dislike> <Eve> .
<Bob> <focus> <Alice> .
<Bob> <like> <Eve> .
<Carol> <focus> <Bob> .
<Carol> <like> <Bob> .
<Carol> <dislike> <Francis> .
<Dave> <focus> <Alice> .
<Dave> <focus> <Eve> .
<Dave> <dislike> <Francis> .
<Eve> <like> <Carol> .
<Francis> <like> <Carol> .
<Francis> <dislike> <Dave> .
<Francis> <dislike> <Eve> .
```

The above data are illustrated below:



Unless otherwise specified, functions that return paths represent a path/a ring/a subgraph in JSON format as follows:

```
{
    "src": "<src_IRI>", "dst": "<dst_IRI>",
    "edges": [
        { "fromNode": 0, "toNode": 1, "predIRI": "<pred>" }
    ],
    "nodes": [
        { "nodeIndex": 0, "nodeIRI": "<src_IRI>" },
        { "nodeIndex": 1, "nodeIRI": "<dst_IRI>" }
    ]
}
```

The final return value represents a set of paths/rings/subgraphs as follows :(where the 'paths' element has the format above)

```
{ "paths": [{...}, {...}, ...] }
```

6.6.2 Path-related query

(1) The cycle detection query

Queries for the existence of a cycle containing nodes `u` and `v`

```
cyclePath(u, v, directed, pred_set)
cycleBoolean(u, v, directed, pred_set)
```

Used in SELECT statements, using the same syntax as aggregate functions

Parameter

`u`, `v` : Variable or node IRI

`directed` : A Boolean value, true for directed and false for undirected (all edges in the graph are considered bidirectional

`pred_set` : The set of predicates that make up the edges of a ring. If set to null '{}', all predicates in the data are allowed

Return value

- `cyclePath` : Returns a ring containing the nodes `u` and `v` (if any) in JSON form. If `u` or `v` is a variable, a loop is returned for each set of valid values of the variable
- `cycleBoolean` : Return true if there is a ring containing nodes `u` and `v`; Otherwise, return false

The following query asks if there is a directed ring that contains Carol, a person Francis dislikes (Dave or Eve in the sample data), and whose edges can only be marked by a "like" relationship:

```
select (cycleBoolean(?x, <Carol>, true, {<like>}) as ?y)where{  <Francis> <dislike> ?x .}
```

Results are as follows:

?y
"true"^^<http://www.w3.org/2001/XMLSchema#>

If you want to output a loop that meets the above criteria, use the following query:

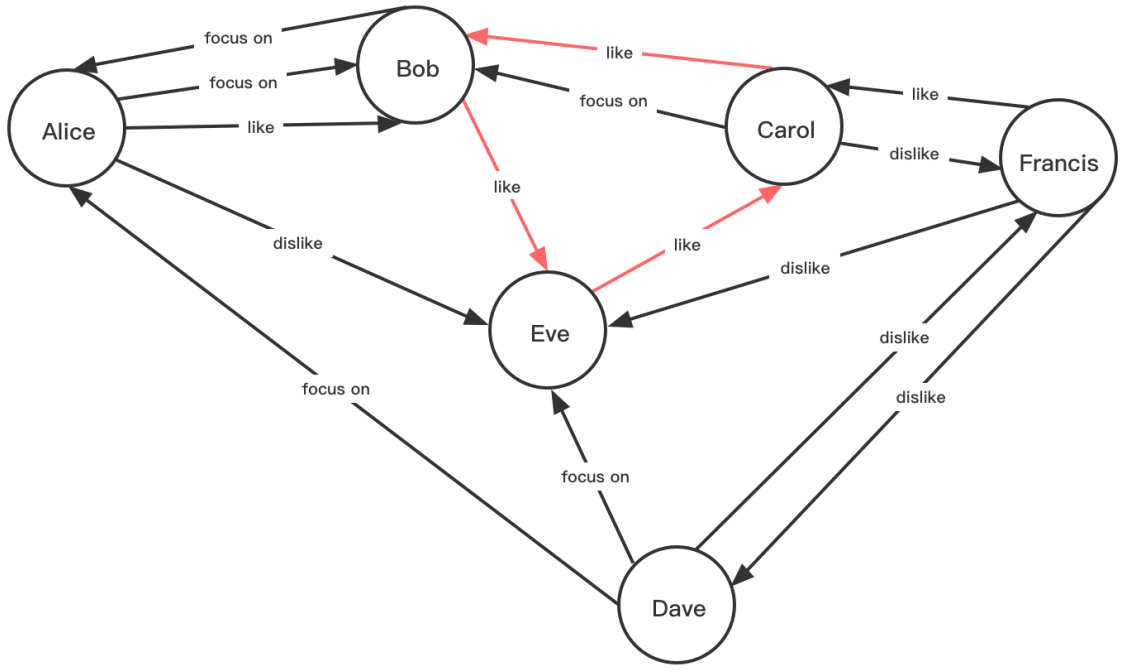
```
SELECT (cyclePath(?x, <Carol>, true, {<like>}) as ?y)
WHERE
{
    <Francis> <dislike> ?x .
}
```

Results are as follows, It can be seen that one of the rings satisfying the condition is formed by Eve likes carol-carol likes bob-bob likes Eve in sequence :(the escape of the outermost double quotation marks and the inner double quotation marks are omitted for ease of reading)

```
{
    "paths": [
        {
            "src": "<Eve>",
            "dst": "<Carol>",
            "edges": [
                {
                    "fromNode": 2,
                    "toNode": 3,
                    "predIRI": "<like>"
                },
                {
                    "fromNode": 3,
                    "toNode": 1,
                    "predIRI": "<like>"
                },
                {
                    "fromNode": 1,
                    "toNode": 2,
                    "predIRI": "<like>"
                }
```

```
            ],
            "nodes": [
                {
                    "nodeIndex": 1,
                    "nodeIRI": "<Bob>"
                },
                {
                    "nodeIndex": 3,
                    "nodeIRI": "<Carol>"
                },
                {
                    "nodeIndex": 2,
                    "nodeIRI": "<Eve>"
                }
            ]
        }
    ]
}
```

The red part below is the ring:



(2) Shortest path Query

Query the shortest path from node `u` to node `v`

```
shortestPath(u, v, directed, pred_set)
shortestPathLen(u, v, directed, pred_set)
```

Used in SELECT statements, using the same syntax as aggregate function.

Parameter

`u` , `v` : Variable or node IRI

`directed` : Boolean value, true for directed, false for undirected (all edges in the graph are considered bidirectional)

`pred_set` : The set of predicates that are allowed to occur on the side that makes up the shortest path. If set to null '{}', all predicates in the data are allowed

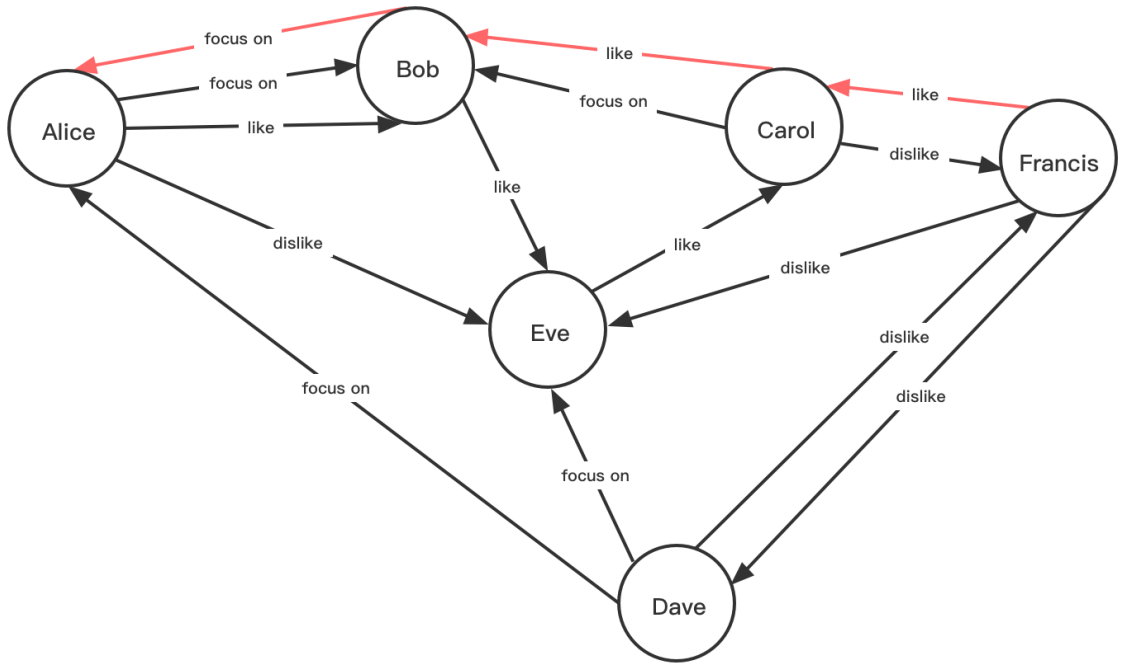
Return value

- `shortestPath` : Returns a shortest path (if reachable) from node `u` to `v` in JSON form. If `u` or `v` is a variable, a shortest path is returned for each set of valid values of the variable.
- `shortestPathLen` : Returns the shortest path length (if reachable) from node `u` to `v`. If `u` or `v` is a variable, return a shortest path length value for each set of valid values of the variable。

The following query returns the shortest path from Francis to a person (Alice in the example data) that Bob likes, cares about, or dislikes, and is not disliked by Francis, with a relationship that can be like or care about.

```
SELECT (shortestPath(<Francis>, ?x, true, {<like>, <focus>}) AS ?y)
WHERE{  <Bob> ?pred ?x .
      MINUS { <Francis> <dislike> ?x . }
     }
```

The red part below is the shortest path:



Results are as follows：(For easy reading, the outermost double quotation mark and the escape of the inner double quotation mark are omitted.)

```
{
    "paths": [
        {
            "src": "<Francis>",
            "dst": "<Alice>",
            "edges": [
                {
                    "fromNode": 4,
                    "toNode": 3,
                    "predIRI": "<like>"
                },
```

```
                {
                    "fromNode": 3,
                    "toNode": 1,
                    "predIRI": "<like>"
                },
                {
                    "fromNode": 1,
                    "toNode": 0,
                    "predIRI": "<focus>"
                }
            ],
            "nodes": [
                {
                    "nodeIndex": 0,
                    "nodeIRI": "<Alice>"
                },
                {
                    "nodeIndex": 1,
                    "nodeIRI": "<Bob>"
                },
                {
                    "nodeIndex": 3,
                    "nodeIRI": "<Carol>"
                },
                {
                    "nodeIndex": 4,
                    "nodeIRI": "<Francis>"
                }
            ]
        }
    ]
}
```


If you want to output only the shortest path length, use the following query:

```
SELECT (shortestPathLen(<Francis>, ?x, true, {<like>, <focus>}) AS ?y)
WHERE{  <Bob> ?pred ?x .
      MINUS { <Francis> <dislike> ?x .}
     }
```

Results are as follows：(For easy reading, the outermost double quotation mark and the escape of the inner double quotation mark are omitted)


```
{
    "paths": [
        {
            "src": "<Francis>",
            "dst": "<Alice>",
            "length": 3
        }
    ]
}
```


(3) Single-Source Shortest Path

Queries the shortest path from a source node `u` to all other nodes.

```
SSSP(u, directed, pred_set)
SSSPLen(u, directed, pred_set)
```

Parameters

`u`: A variable or node IRI representing the source node.

`directed`: A boolean value; if true, the graph is directed; if false, all edges in the graph are considered bidirectional.

`pred_set`: A set of predicates to consider. If set to `{}`, all predicates in the data are allowed.

Return Values

`SSSP` returns the shortest paths. The return value is in the following format, where `src` is the IRI corresponding to `u`; `dst` is the IRI corresponding to a reachable node; `nodes` contains the indices and IRIs of the nodes involved in the path; `edges` contains the start and end node indices and the predicate IRI involved in the path.

```
{
    "paths": [
        {
            "src": "<src_IRI>",
            "dst": "<dst_IRI>",
            "edges": [
                {
                    "fromNode": 0,
                    "toNode": 1,
                    "predIRI": "<pred>"
                }
            ],
            "nodes": [
                {
                    "nodeIndex": 0,
                    "nodeIRI": "<src_IRI>"
                },
                {
                    "nodeIndex": 1,
                    "nodeIRI": "<dst_IRI>"
                }
            ]
        },
        ...
    ]
}
```

`SSSPLen` returns the shortest path lengths. The return value is in the following format, where `src` is the IRI corresponding to `u`; `dst` is the IRI corresponding to a reachable node; `length` is the shortest path length from `src` to `dst`.

```
{
    "paths": [
        {
            "src": "<src_IRI>",
            "dst": "<dst_IRI>",
            "length": 0
        },
        ...
    ]
}
```

(4) Reachability/K-hop reachability query

Query whether node `u` is reachable or K-hop reachable to node `v` (i.e., there exists a path with `u` as its source and `v` as its destination whose length does not exceed `k`).

```
kHopReachable(u, v, directed, k, pred_set)kHopReachablePath(u, v, directed, k, pred_set)
```

Parameter

`u`, `v` : Variable or node IRI

`k` : If it is set to a non-negative integer, it is the upper limit of the path length (query k-hop reachability). If set to negative, query reachability

`directed` : A Boolean value, true for directed and false for undirected (all edges in the graph are considered bidirectional

`pred_set` : The set of predicates that are allowed to occur on the side that constitutes a path. If set to null `{}`, all predicates in the data are allowed

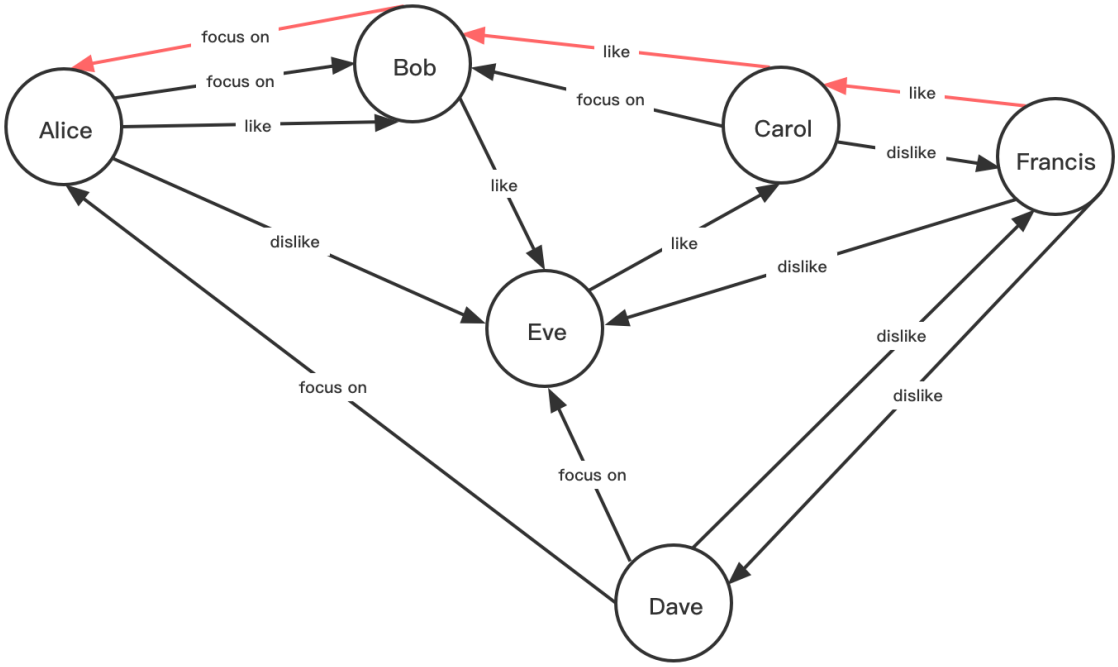
Return value

- `kHopReachable`: Return true if node `u` is reachable to node `v` (or K hop reachable, depending on the value of parameter `K`); Otherwise, return false. If `u` or `v` is a variable, return a true/false value for each set of valid values of the variable
- `kHopReachablePath`: Returns any path from node `u` to node `v` (if reachable) or a k-hop path, that is, a path with length less than or equal to `K` (if reachable, depending on the value of `K`). If `u` or `v` is a variable, return a path (if reachable) or a k-hop path (if reachable) for each set of valid values of the variable

The following query follows the example query from the previous section, "Shortest path Query" : It starts with Francis and ends with a person that Bob likes, cares about, or dislikes, and is not disliked by Francis (which is Alice in the example data). Ask if the relationship between the two people is 2 hops or within reach through liking or following.

```
SELECT (kHopReachable(<Francis>, ?x, true, 2, {<like>, <focus>}) AS ?y)
WHERE{  <Bob> ?pred ?x .
      MINUS { <Francis> <dislike> ?x . }
     }
```

Since the shortest path length satisfying the condition is known to be 3:



Therefore, the above query results are false:

```
{"paths":[{"src":"<Francis>","dst":"<Alice>","value":"false"}]}
```

Francis and Alice, on the other hand, are reachable, but the shortest path length exceeds the above limit. So a query for reachability (with `k` set to negative) returns true:

```
SELECT (kHopReachable(<Francis>, ?x, true, -1, {<like>, <focus>}) AS ?y)
WHERE{  <Bob> ?pred ?x .
      MINUS { <Francis> <dislike> ?x . }
     }
```

Results are as follows:

```
{"paths":[{"src":"<Francis>","dst":"<Alice>","value":"true"}]}
```

If you want to return a path that satisfies the condition between two people, you can call the `kHopReachablePath` function:

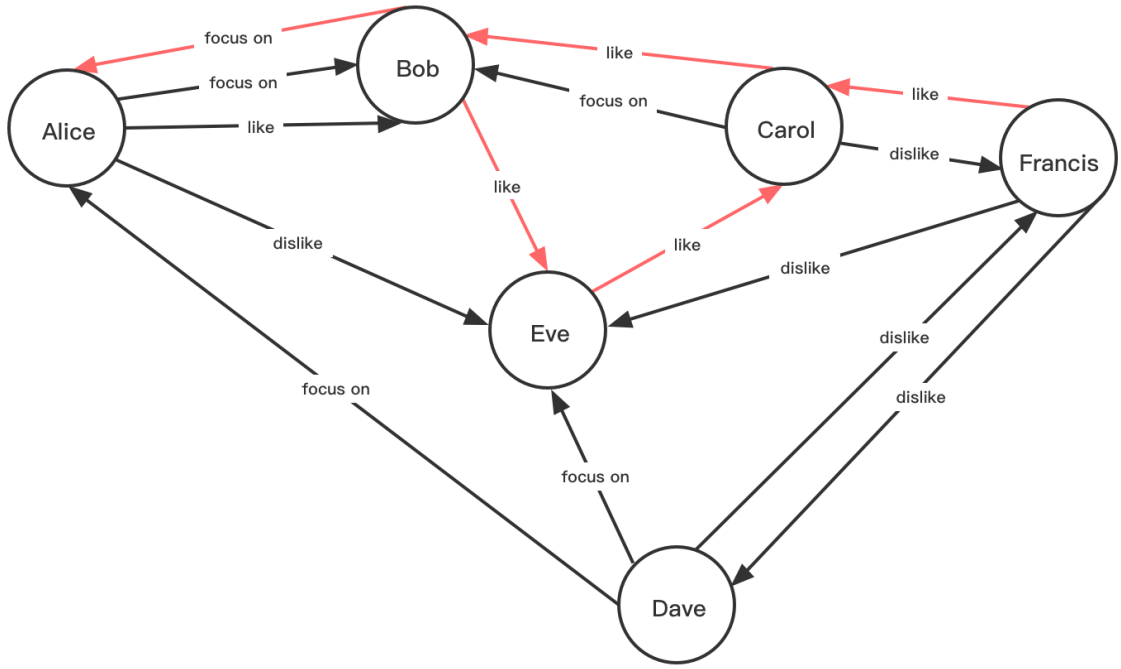
```
SELECT (kHopReachablePath(<Francis>, ?x, true, -1, {<like>, <focus>}) AS ?y)
WHERE{  <Bob> ?pred ?x .
      MINUS { <Francis> <dislike> ?x . }
     }
```

The result may be the shortest path described above:

```
{
    "paths": [
        {
            "src": "<Francis>",
            "dst": "<Alice>",
            "edges": [
                {
                    "fromNode": 4,
```

```
                    "toNode": 3,
                    "predIRI": "<like>"
                },
                {
                    "fromNode": 3,
                    "toNode": 1,
                    "predIRI": "<like>"
                },
                {
                    "fromNode": 1,
                    "toNode": 0,
                    "predIRI": "<focus>"
                }
            ],
            "nodes": [
                {
                    "nodeIndex": 0,
                    "nodeIRI": "<Alice>"
                },
                {
                    "nodeIndex": 1,
                    "nodeIRI": "<Bob>"
                },
                {
                    "nodeIndex": 3,
                    "nodeIRI": "<Carol>"
                },
                {
                    "nodeIndex": 4,
                    "nodeIRI": "<Francis>"
                }
            ]
        }
    ]
}
```


It could also be a non-shortest path with a ring in it, as shown in the figure below:



(5) All K-hop paths

Query all K-hop reachable paths from node `u` to node `v`.

```
kHopEnumerate(u, v, directed, k, pred_set)
```

Parameter

`u`, `v` : Variable or node IRI

`k` : if set to a non-negative integer, it is the upper limit of the path length (querying K-hop reachability); if set to a negative number, it queries reachability

`directed` : Boolean value, true indicates directed, false indicates undirected (all edges in the graph are considered bidirectional)

`pred_set` : The set of predicates allowed on the edges that form the path. If set to an empty `{}`, it means that all predicates in the data are allowed.

Return Value

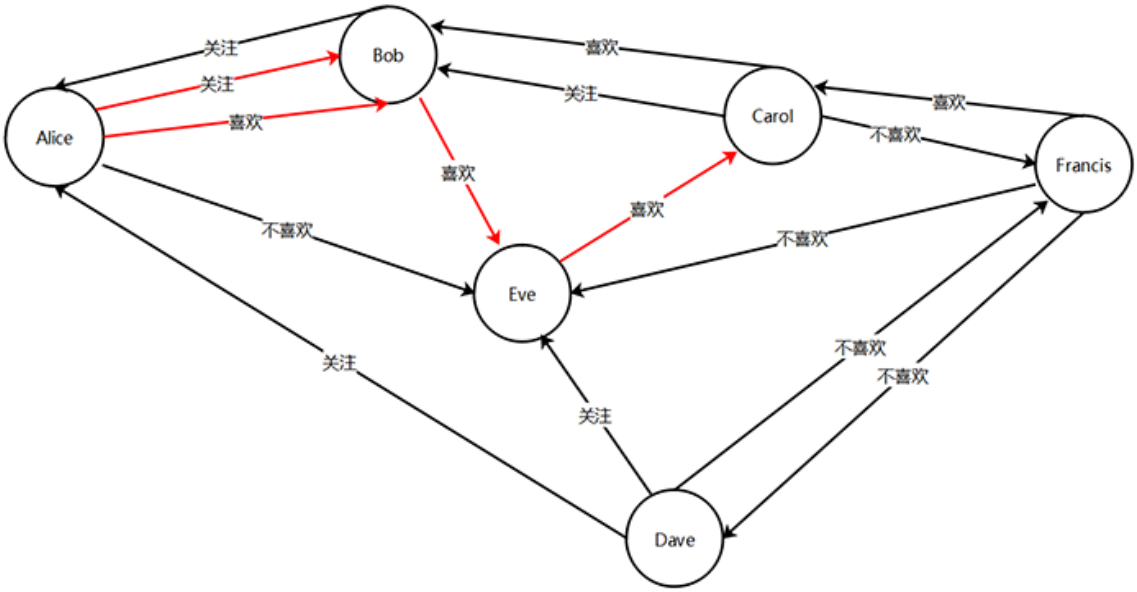
Return all paths (if reachable) or K-hop paths (if K-hop reachable, depending on the value of parameter `k`) from node `u` to node `v`. If `u` or `v` is a variable, return all paths (if reachable) or K-hop paths (if K-hop reachable) for each valid value of the variable.

Example

The return value is in the following form, where SRC is the result of IRI or variable query corresponding to u. Which destination nodes DST contains depends on the second argument to the function; The corresponding PPR value is a double precision floating point number.

```
SELECT (kHopEnumerate(<Alice>, ?x, true, 3, {<like>, <focus>}) AS ?y)
WHERE
{
    <Francis> ?pred ?x.
    MINUS { <Alice> <dislike> }
    }
```

Query the path between Alice and Francis, who is liked, followed, or disliked by Francis and not disliked by Alice. The path should be reachable within 3 hops through liking or following relationship. (Example data: Carol)




```
{
    "paths": [
        {
            "src": "<Alice>",
            "dst": "<Carol>",
            "edges": [
                {
                    "fromNode": 0,
                    "toNode": 1,
                    "predIRI": "<like>"
                },
                {
                    "fromNode": 1,
                    "toNode": 2,
                    "predIRI": "<like>"
                },
                {
                    "fromNode": 2,
                    "toNode": 3,
                    "predIRI": "<like>"
                }
            ],
            "nodes": [
                {
                    "nodeIndex": 3,
                    "nodeIRI": "<Carol>"
                },
                {
                    "nodeIndex": 2,
                    "nodeIRI": "<Eve>"
                },
                {
                    "nodeIndex": 0,
                    "nodeIRI": "<Alice>"
                },
                {
                    "nodeIndex": 1,
```

```
                    "nodeIRI": "<Bob>"
                }
            ]
        },
        {
            "src": "<Alice>",
            "dst": "<Carol>",
            "edges": [
                {
                    "fromNode": 0,
                    "toNode": 1,
                    "predIRI": "<focus>"
                },
                {
                    "fromNode": 1,
                    "toNode": 2,
                    "predIRI": "<like>"
                },
                {
                    "fromNode": 2,
                    "toNode": 3,
                    "predIRI": "<like>"
                }
            ],
            "nodes": [
                {
                    "nodeIndex": 3,
                    "nodeIRI": "<Carol>"
                },
                {
                    "nodeIndex": 2,
                    "nodeIRI": "<Eve>"
                },
                {
                    "nodeIndex": 0,
                    "nodeIRI": "<Alice>"
                },
                {
                    "nodeIndex": 1,
                    "nodeIRI": "<Bob>"
                }
            ]
        }
    ]
}
```


(6) kHopCount

Query the number of nodes accessible from node `u` within k layers.

```
kHopCount(u, directed, k, pred_set)
```

Parameter

`u`: variable or node IRI
`k`: Number of hops (only counting the nodes reachable within this number of hops)
`directed`: Boolean value, true for directed, false for undirected (all edges in the graph are considered bidirectional)
`pred_set`: The set of predicates that are allowed to occur on the edges that makes up the shortest path. If set to null `{}`, all predicates in the data are allowed.

Return value

The return value is in the following format, where src is the IRI corresponding to u; depth is the level/height at which the node is located (equal to the parameter k); and count is the total number of nodes visited at the current level, with a type of integer.

```
{
    "paths":[
        { "src": "<Alice>", "depth": 3, "count": 1}
    ]
}
```

(7) KHopNeighbor

Query the nodes that are reachable from node u within k layers.

```
kHopNeighbor(u, directed, k, pred_set, ret_num)
```

Parameter

`u`: variable or node IRI
`k`: Number of hops (only counting the nodes reachable within this number of hops)
`directed`: Boolean value, true for directed, false for undirected (all edges in the graph are considered bidirectional)
`ret_num`: Integer, optional, defaulted to 100, representing the maximum number of node IRIs to return. If the total number of nodes is less than ret_num, all node IRIs will be returned.

Return value:

The return value is in the following format, where src is the IRI corresponding to `u`; depth is the level/height at which the node is located (equal to the parameter `k`); and dst is a list of nodes visited at the current level.

```
{
    "paths":[
        {
            "src": "<Alice>",
            "depth": 3,
            "dst": [
                "<Car>"
            ]
        }
    ]
}
```

(8) bfsCount

The query starts from node u and outputs the number of nodes accessed in different layers in the breadth-first traversal order.

```
bfsCount(u, directed, pred_set)
```

Parameter

`u`: variable or node IRI, indicating the source node
`directed`: boolean value, true for directed, false for undirected (all edges in the graph are considered bidirectional)
`pred_set`: The set of predicates that are allowed to occur on the side that makes up the shortest path. If set to null `{}`, all predicates in the data are allowed.

Return value

The return value is in the following form, where src is the IRI corresponding to u; depth is the number of layers/height (only accesses u when depth is 0); count is the total number of nodes accessed at the layer, of integer type.

```
{"paths":
    [
        {"src":"<Alice>",
        "results":[{"depth":0, "count":1}, ...]
        }
    ]
}
```

Sample

The following query returns the directed breadth-first traversal count with Alice as the source node. The relationship on the edge can be like, follow, or dislike, and the query is:

```
SELECT(bfsCount(<Alice>,true,{<like>,<focus>,<dislike>}) AS ?y)
WHERE{}
```

The results are as follows:

```
{"paths":
    [
        {"src":"<Alice>",
         "results":[{"depth":0, "count":1}, {"depth":1, "count":2}, {"depth":2,
"count":1}, {"depth":3, "count":1}, {"depth":4, "count":1}]
        }
    ]
}
```

(9) diameterEstimation

Diameter estimation algorithm, returns the length of the longest shortest path.

```
diameterEstimation(pred_set)
```

Parameters

`pred_set`: The set of predicates to consider (if set to an empty `{}`, it indicates that all predicates in the data are allowed).

Example:

The following query calculates the diameter based on the relationships "likes", "follows", or "dislikes"::

```
SELECT (diameterEstimation({<likes>, <follows>, <dislikes>}) AS ?y)WHERE{}
```

Return Value

The return value is in the following format:

```
{
    "paths": [4]
}
```

6.6.3 Importance analysis

(1) PageRank

```
PR(directed, pred_set, alpha, maxIter, tol)
```

Parameter

`directed`: Boolean value, true indicating directed, false indicating undirected (all edges in the graph are considered bidirectional)

`pred_set`: The set of predicates to be considered (if set to an empty `{}`, it means allowing all predicates in the data)

`alpha`: The damping factor of PageRank, representing the probability of following an edge in a random walk model

`maxIter`: The maximum number of iterations for solving PageRank using an iterative method

`tol`: The tolerance level for the error between two iterations in the PageRank value

Return value

The return value is in the following format, where src is the IRI of the node; result is the PageRank value of the corresponding node in the graph, with a type of float.

```
{
    "paths": [
        {
            "src": "<Alice>",
            "results": 0.1
        }
    ]
}
```

(2) Personalized PageRank

```
PPR(u, hopCnt, pred_set, retNum)
```

The FORA algorithm [1] is invoked to calculate the top-K PPR value relative to u.

[1] S. Wang, R. Yang, X. Xiao, Z. Wei, and Y. Yang, “FORA: Simple and Effective Approximate Single Source Personalized PageRank,” in Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax NS Canada, Aug. 2017, pp. 505–514. doi: 10.1145/3097983.3098072.

Parameter

`u`: variable or node IRI, indicating the source node
`hopCnt`: an integer. When it's -1, PPR can be calculated without limitation. Otherwise, it is restricted to `hopCnt` hops
`pred_set`: The set of predicates that are allowed to occur on the side that makes up the shortest path. If set to null '{}', all predicates in the data are allowed

`retNum`: an integer, indicating that the `retNum` node IRIs with the largest PPR and their corresponding PPR values need to be returned (If the total number of nodes in the graph is fewer than `retNum`, then return all node IRIs and their corresponding PPR values)

Return value

The return value is in the following form, where src is u's IRI, or its query result if u is a variable; which destination nodes dst contains depends on the last argument of the function; the PPR value is a double-precision floating point number.

(Note: Since FORA is an approximate algorithm with randomness, it is normal for the return value to be slightly different each time.)

```
{"paths":
    [
        {"src":"<Francis>", "results":
            [{"dst":"<Alice>", "PPR":0.1}, {"dst":"<Bob>", "PPR": 0.01}, ...]
        }, ...
    ]
}
```

Sample

Return the three people (and their PPR values) with top-3 PPR values corresponding to all the people Bob likes, follows, or dislikes:

```
select (PPR(?x, -1, {}, 3) as ?y)
where
{
    <Bob> ?pred ?x .
}
```

(3) closenessCentrality

Query how easy it is for a node to reach other nodes.

```
closenessCentrality(u, directed, pred_set)
```

Parameter

`u`: variable or node IRI, indicating the source node
`directed`: boolean value, true for directed, false for undirected (all edges in the graph are considered bidirectional)
`pred_set`: The set of predicates that are allowed to occur on the side that makes up the shortest path. If set to null `{}`, all predicates in the data are allowed.

Return value

The return value is in the following form, where src is the IRI corresponding to u; result is the closeness centrality of node u in the graph and is of double-precision floating-point type.

```
{
    "paths": [
        {
            "src": "<Alice>",
            "result": 0.5
        }
    ]
}
```

Sample

Example 1. The query returns the closeness centrality of **Alice** in an **undirected graph** (all edges in the graph are regarded as bidirectional), and the relationship between edges can be **like** or **follow**. The SPARQL query is:

```
SELECT (closenessCentrality(<Alice>, false, {<like>, <focus>}) AS ?x) WHERE{}
```

The result is as follows: (For the convenience of reading, the escaping of the outermost double quotes and the inner double quotes of the string is omitted)

```
{
    "paths":[
        {
            "src": "<Alice>",
            "result": 0.555556
        }
    ]
}
```

In the above query, the shortest distance for Alice to reach the other nodes is as follows, and the average distance can be calculated as 1.8, and the value of closeness centrality is 1/1.8 =0.555556, which is consistent with the result.

```
{
    "Bob" : 1,
    "Dave" : 1,
    "Eve" : 2,
    "Carol" : 2,
    "Francis" : 3
}
```

Example 2. The query returns **Alice**'s closeness centrality in the **directed graph**, and the relationship of edges can be **like** or **follow**. The SPARQL query is:

```
SELECT (closenessCentrality(<Alice>, true, {<like>, <focus>}) AS ?x) WHERE{}
```

The result is as follows: (For the convenience of reading, the escaping of the outermost double quotes and the inner double quotes of the string is omitted)

```
{
    "paths":[
        {
            "src": "<Alice>",
            "result": 0.500000
        }
    ]
}
```

In the above query, the shortest distance for Alice to reach the other nodes is as follows, and the average distance can be calculated as 2, and the value of closeness centrality is 1/2 =0.5, which is consistent with the result.

```
{
    "Bob" : 1,
    "Eve" : 2,
    "Carol" : 3
}
```

(4) triangleCounting

Count the number of triangles in the graph.

```
triangleCounting(directed, pred_set)
```

Parameter

`u`: variable or node IRI, indicating the source node
`directed`: boolean value, true for directed, false for undirected (all edges in the graph are considered bidirectional)
`pred_set`: The set of predicates that are allowed to occur on the side that makes up the shortest path. If set to null `{}`, all predicates in the data are allowed.

Return value

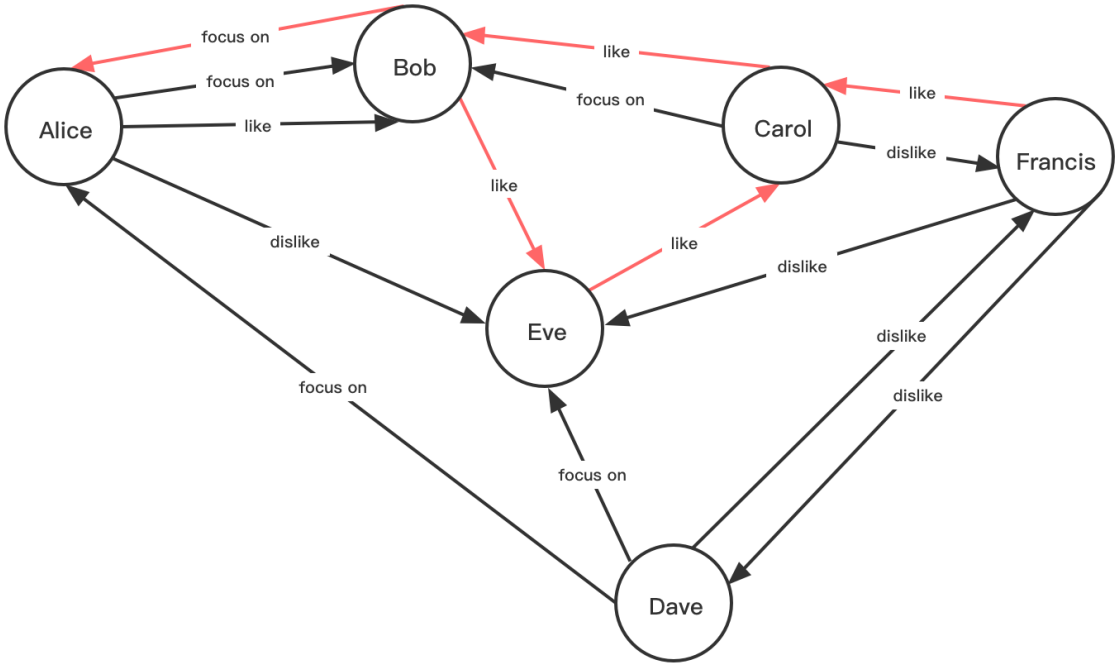
The return value is in the following form:

```
{
    "paths": [2]
}
```

Sample

Query the number of directed triangles in the graph, and the edges that form it can only be labeled by the **like** relation. The SPARQL query statement is:

```
select (triangleCounting(true, {<like>}) as ?y) where {}
```



The result is as follows. The number of directed triangles in the graph whose sides can only be marked by the "like" relation is 1, that is, Bob -> Eve -> Carol-> Bob:


```
{
    "paths": [1]
}
```

(5) betweennessCentrality

Betweenness Centrality algorithm.

```
betweennessCentrality(uid, directed, pred_set)
```

Parameters

`uid`: Variable or node IRI

`directed`: Boolean, true for directed, false for undirected (all edges in the graph are treated as bidirectional)

`pred_set`: The set of predicates to consider (if set to `{}`, it means all predicates in the data are allowed)

Example:

The following query calculates the betweenness centrality for the node Eve, considering the relationships "likes", "follows", and "dislikes" as edges:

```
SELECT (betweennessCentrality(<Eve>, true, {<likes>, <follows>, <dislikes>}) AS
?y) WHERE{}
```

Return value

The return value is in the following format:

```
{
    "paths":[
        {"src":"<Eve>", "result":6.5}
    ]
}
```

6.6.4 Community detection analysis query

(1) Weakly Connected Components

Return all weakly connected components of the graph.

```
WCC(pred_set)
```

Parameters

`pred_set`: The set of predicates allowed on the edges that form the weakly connected components. If set to an empty `{}`, it means that all predicates in the data are allowed to appear.

Return value

Nested arrays, with the same form as the return value of label propagation.

(2) tag propagation

Based on tag propagation, it is possible to query the clustering status of each node in a graph and apply it to various applications such as community detection.

```
labelProp(directed, pred_set)
```

Parameter

`directed`: Boolean value, true indicating directed, false indicating undirected (all edges in the graph are considered bidirectional)

`pred_set`: The set of predicates to consider (if set to an empty `{}`, it means allowing all predicates in the data)

`maxIter`: Maximum iteration times

Return value

The returned value is an array of arrays (nested arrays), where the elements are node IRIs, corresponding to a partition of the nodes in the graph.

```
{
    "paths": [
        [
            "<Alice>",
            "<Bob>"
        ],
        [
            "<Carol>"
        ]
    ]
}
```

(3) Louvain

Louvain Community Discovery algorithm, by constantly merging point communities to maximize the modularity of the graph, can discover the hierarchical community structure

```
louvain(directed, pred_set, maxIter, increase)
```

Parameter:

`directed`: Boolean value, true indicating directed, false indicating undirected (all edges in the graph are considered bidirectional).

`pred_set`: The set of predicates to consider (if set to an empty {} , it means that all predicates in the data are allowed to appear).

`maxIter`: The maximum number of iterations in the first stage (>=1).

`increase`: Threshold for the gain in modularity (0~1).

Return value:

The return value is in the following format: count is the number of communities divided, details is the information of each divided community, including community ID (communityId) and member number (memberNum).

```
{
    "count": 3,
    "details": [
        { "communityId": "2", "menberNum": 5},
        { "communityId": "4", "menberNum": 5},
        { "communityId": "5", "menberNum": 4}
    ]
}
```

6.6.5 Correlation analysis query

(1) Local Agglomeration Coefficient

Query the local clustering coefficient of node u, which is the number of edges between all nodes connected to it (i.e., the number of triangles formed with u as the vertex), divided by the maximum number of edges that can be connected between these nodes (i.e., the maximum number of triangles that can be formed with u as the vertex).

```
clusterCoeff(u, directed, pred_set)
```

Parameters

`u`: variable or node IRI

`directed`: Boolean value, true indicates directed, false indicates undirected (all edges in the graph are considered bidirectional). When the graph is considered directed, only cycle-type triangles are counted (see the introduction to triangle counting for details).

`pred_set`: the set of predicates considered (if set to an empty `{}`, it means that all predicates in the data are allowed to appear)

Return value

The return value is the local clustering coefficient of node u, and the corresponding value is a double-precision floating-point number (see the following example for details).

(2) Overall agglomeration coefficient

Query the overall clustering coefficient of the graph.

```
clusterCoeff(directed, pred_set)
```

Parameters

`directed`: Boolean value, true indicates directed, false indicates undirected (all edges in the graph are considered bidirectional). When the graph is considered directed, only cycle-type triangles are counted (see the introduction to triangle counting for details).

`pred_set`: the set of predicates considered (if set to an empty `{}`, it means that all predicates in the data are allowed to appear)

Return value

The return value is the overall clustering coefficient of the graph, and the corresponding value is a double-precision floating-point number.

(3) JaccardSimilarity

Calculate the Jaccard coefficient between point pairs to compare the similarity between points.

```
JaccardSimilarity(uid, pred_set, k, retNum)
```

Parameters

`uid`: Variable or node IRI

`pred_set`: The set of predicates to consider (if set to `{}`, it means all predicates in the data are allowed)

`k`: If set to a non-negative integer, it represents the maximum path length (query includes paths within k hops); if set to a negative number, there is no upper limit

`ret_num`: Integer, optional, default is 10, indicating the maximum number of node IRIs to return (if the total number of nodes is less than `ret_num`, return all node IRIs)

Example:

The following query returns the set of nodes most similar to Alice, with a maximum hop count of 2 and a maximum return quantity of 10:

```
SELECT (JaccardSimilarity(<Alice>, {}, 2, 10) AS ?y) WHERE{}
```

Return value

The return value is in the following format, where `dst` is the similar node, and `value` is the similarity score:

```
{
    "paths":[
        {"dst":<Francis>, "value":0.5},
        {"dst":<Carol>, "value":0.5},
        {"dst":<Eve>, "value":0.5},
        {"dst":<Dave>, "value":0.333333},
        {"dst":<Bob>, "value":0.333333}
    ]
}
```

(4) degreeCorrelation

The degree correlation of the graph is calculated by calculating the Pearson coefficient between any pair of adjacent points.

```
degreeCorrelation(uid, k, pred_set)
```

Parameters

`uid`: Variable or node IRI

`k`: If set to a non-negative integer, it represents the maximum path length (query includes paths within k hops); if set to a negative number, there is no upper limit

`pred_set`: The set of predicates to consider (if set to `{}`, it means all predicates in the data are allowed)

Example:

The following query calculates the degree correlation of the subgraph formed by the node Alice:

```
SELECT (degreeCorrelation(<Alice>, -1, {}) AS ?y) WHERE{}
```

Return value

The return value is in the following format:

```
{
    "paths":[
        {"src":"<Alice>", "result":0.0429273}
    ]
}
```

7. gStore Visual Tool Workbench

7.1 Installation and deployment

gStore Workbench is a web tool developed by gStore team for online management of gStore graph database and query visualization of gStore. Currently, gStore official website provides Workbench download, the download link is http://www.gstore.cn. Select [Product] - [gStore Workbench] and after filling in the relevant information, you will get a Workbench package, but you will need to install and deploy. The steps to install and deploy are described in detail below.

- **Download tomcat**

Workbench is a website that requires a Web server as a Web container to run, and we recommend tomcat8 as the Web server, which can be downloaded from https://tomcat.apache.org/download-80.cgi. After downloading the zip package, unzip it.

- **Put the Workbench package in tomcat's webapps directory and unzip it;**
- **Under the bin directory of tomcat**

Start tomcat:

```
[root@node1 bin]# ./startup.sh
```

Stop tomcat:

```
[root@node1 bin]# ./shutdown.sh
```

7.2 Login

7.2.1 Browser Access to the System

Login address is:

```
http://workbench`Self-deployed servers`ip:8080/gworkbench/views/user/login.html
```

7.2.2 Connect to gStore instance

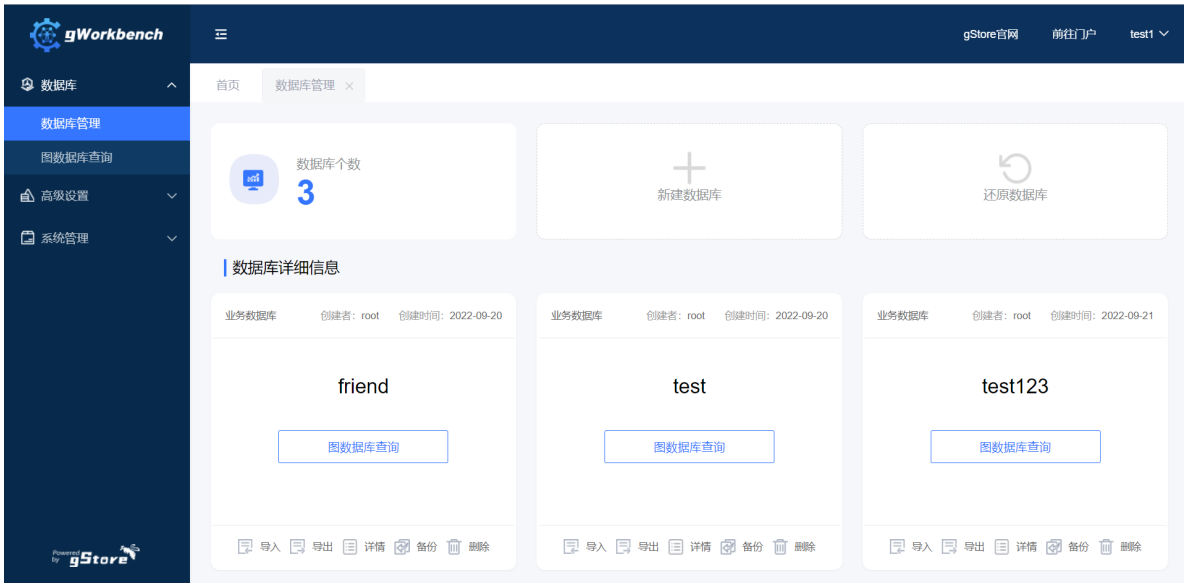
Set the IP address and port of the remote server and save the IP address and port number of the remote server to the gStore diagram database management system. Note that the remote server must install gStore and start the GHTTP service

Enter the user name, password, and verification code to log in to the gStore graph database management system on the saved server (gStore default user name root, password 12345)

7.3 Query functions

7.3.1 Database management

- **View information about the loaded database**



Select the database to view, click the "Details" button below the database, and you will see the specific information of the database.

- **Create a database**

1.Enter the name of the new database, for example, lubm;

2.Upload files in one of the following two ways:
One is to upload from the server. Enter the correct path of the NT file or N3 file. You can enter an absolute path or a relative path; note that a relative path should be relative to the root directory of gStore.

For example, the following two paths are equivalent:

```
/root/gStore/data/lubm.nt    //absolute path
./data/lubm.nt              //relative path
```

The other option is to upload from the local machine. Note that using this approach you must ensure that the **Workbench server is the same server on which gStore is installed**. First select the NT or N3 file locally, then click "Upload File".

3.Click on the "Submit" button.

- **Database deletion**

Click the delete button in the upper right corner of the database and select "Delete" or "Delete Completely" to delete the database. **The system database cannot be deleted.**

- **Import data**

Click [Database Management], select the database to import, click the "Import" icon in the upper right corner; the file type can be server file or local file. To import local files, select nt or N3 format files, click [Upload File], and then click [Import Data].

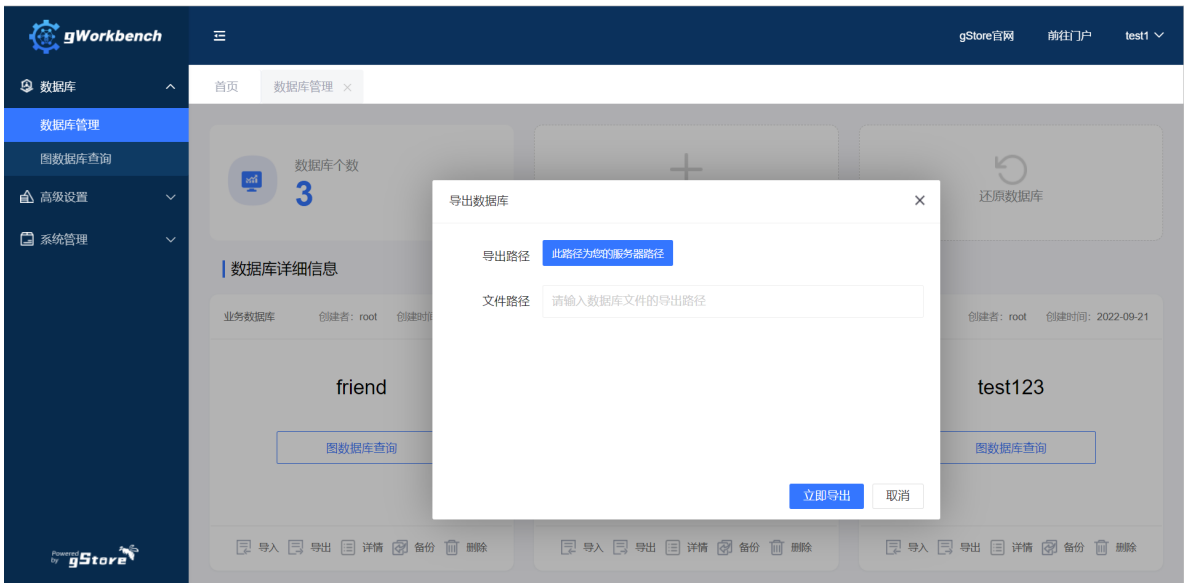
- **Export data**

To export the database as an NT file, click the export button in the lower left corner of the database and select the directory where the exported NT file will reside. You can enter an absolute path or a relative path; note that the current path is the root directory of gStore.

For example, the following two paths are equivalent:

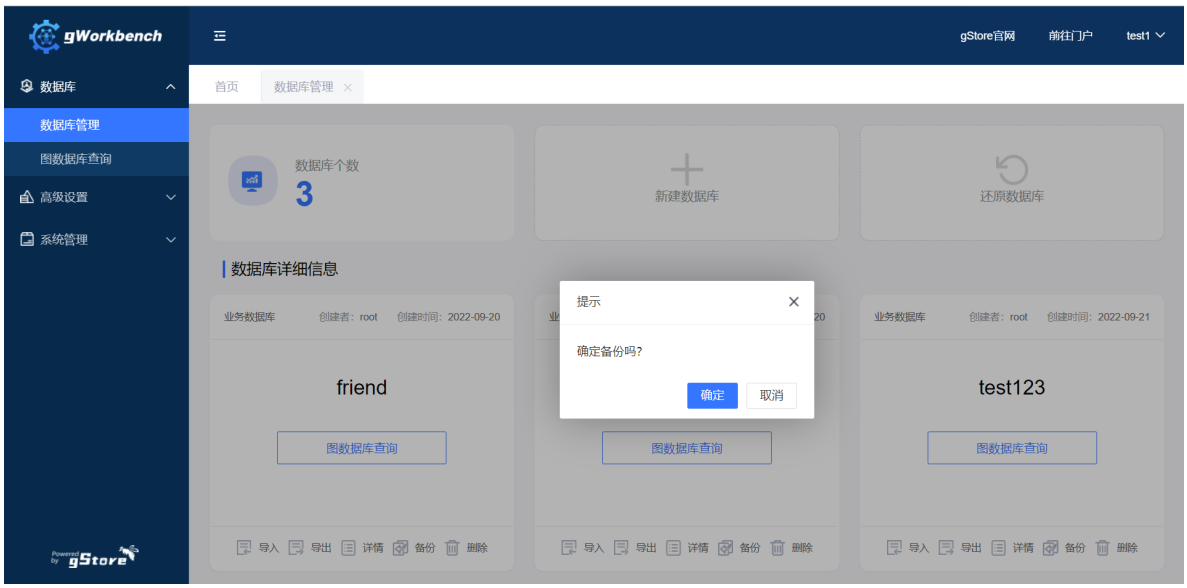
```
/root/gStore/data   //absolute path
./data              //relative path
```

Enter the correct path and click "Export Now". **The system database cannot be exported.**

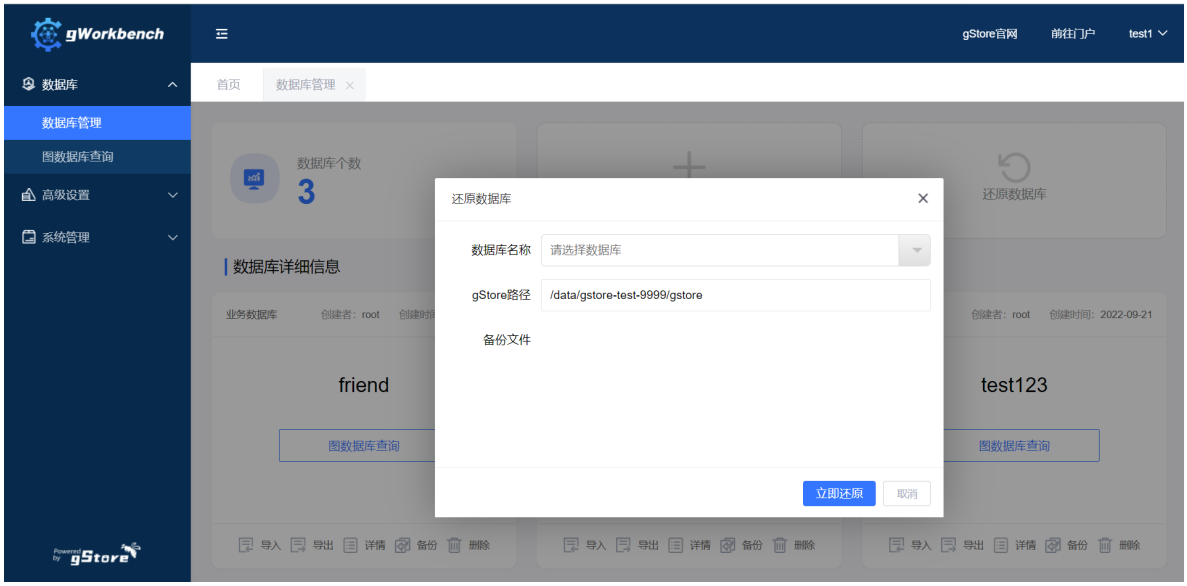


- **Backup database**

Click the [Back up] button on the database you want to back up, and the following dialog box will pop up.



- **Restore database**



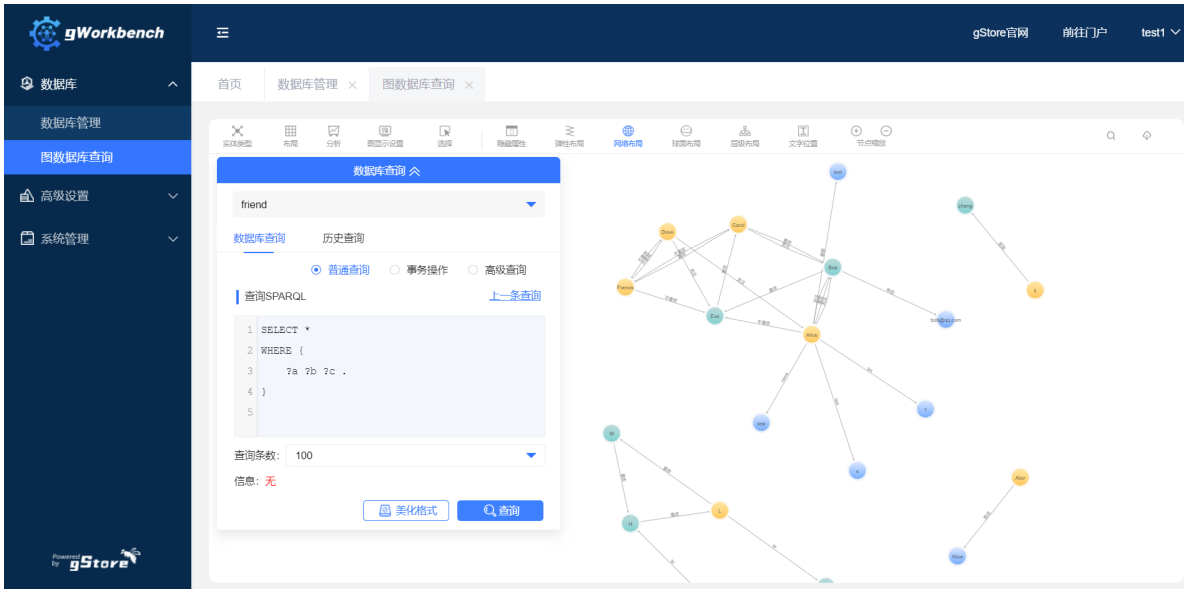
7.3.2 Graph database query

Click [Graph Database query], which contains three functions: ordinary query, transaction operation and advanced query.

(2) Ordinary query

- The default interface is ordinary query. Select the database to query.
- Input the query according to the SPARQL document, then click [Query], and the detailed visual interface of the query results will be displayed on the page.
- After the query, **a menu bar will be displayed, which includes functions such as entity type, layout, analysis, graph display settings, and selection.** Click the download icon in the upper right corner and select JSON list/data list to view the results in the form of json/table.

Graphical display



Json display

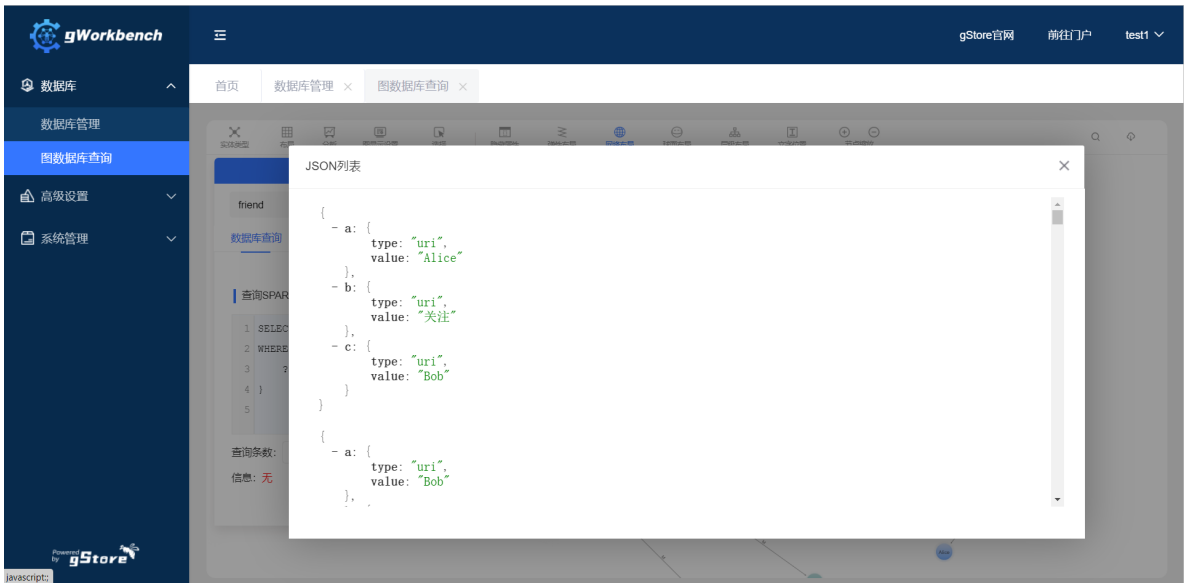
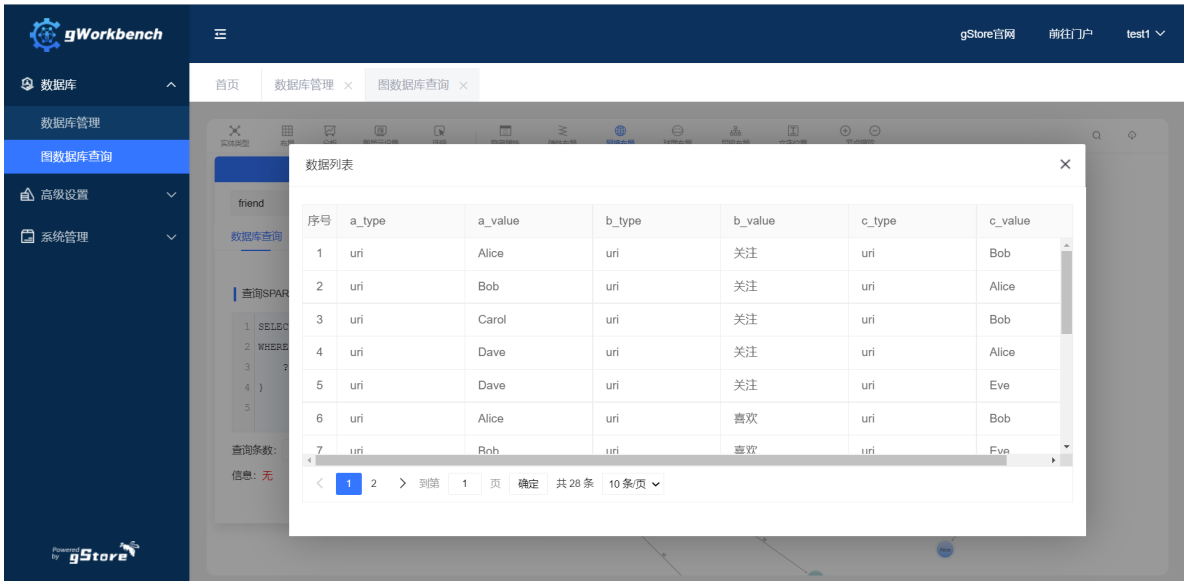


Table display



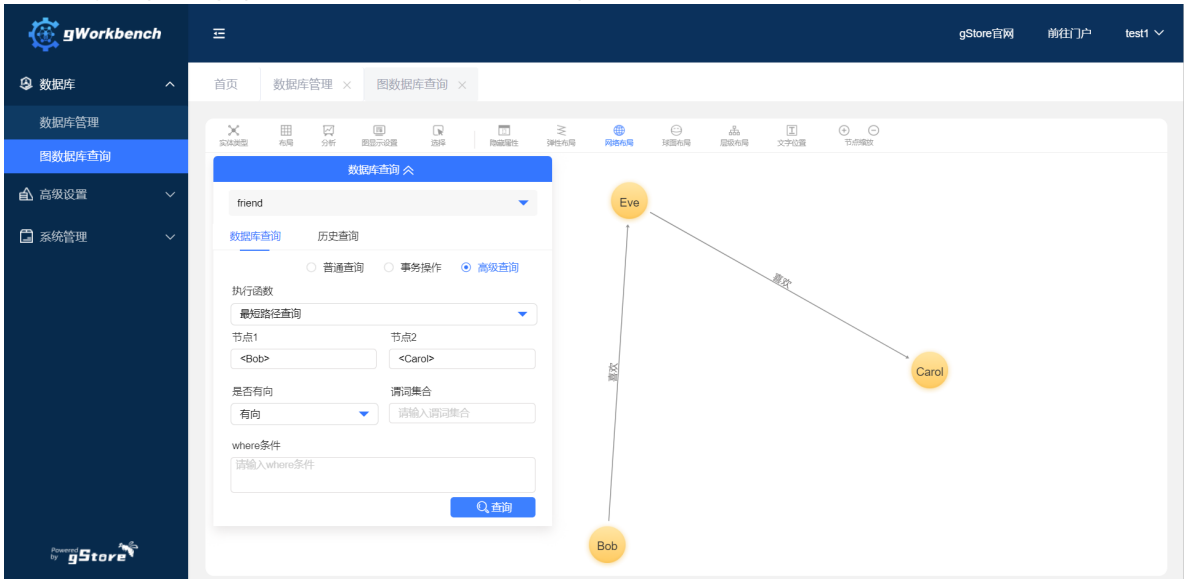
(2) Transaction operation

Click [Transaction Operation], select a database, input the SPARQL statement you want to run in transaction mode, and click [Query].

The insertion and deletion of triples can be achieved by SPARQL statements.

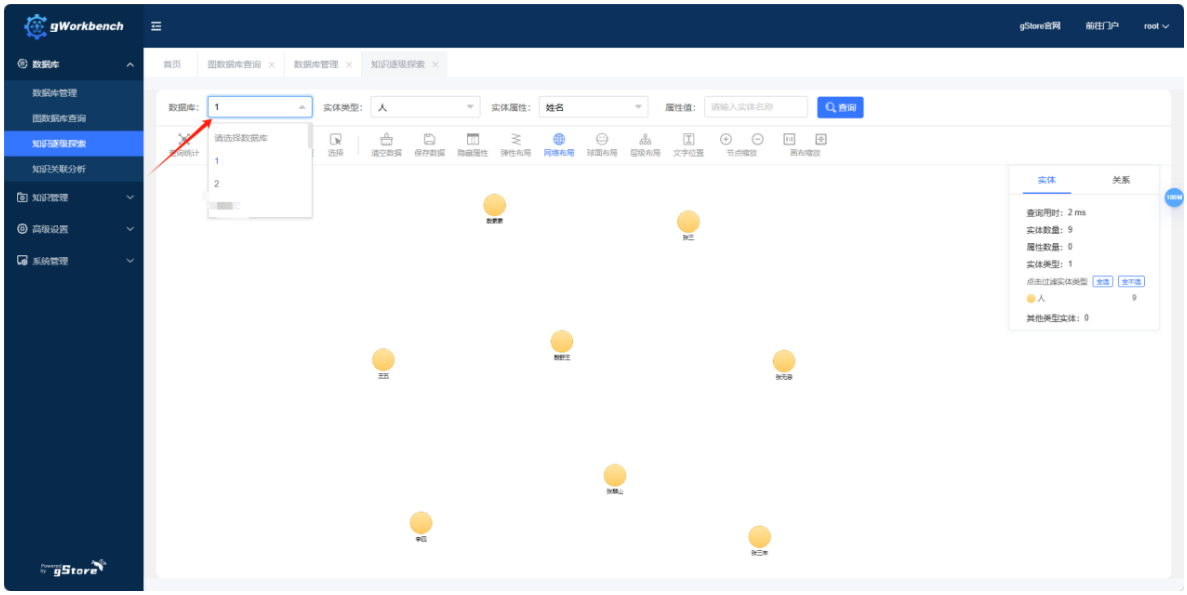
(3) Advanced query

Click [Advanced Query], select a database and a function to execute, fill in other information, and click [Query] to get the corresponding results. The advanced query module can reduce the difficulty of posing queries, since it does not require users to write a full SPARQL statement.

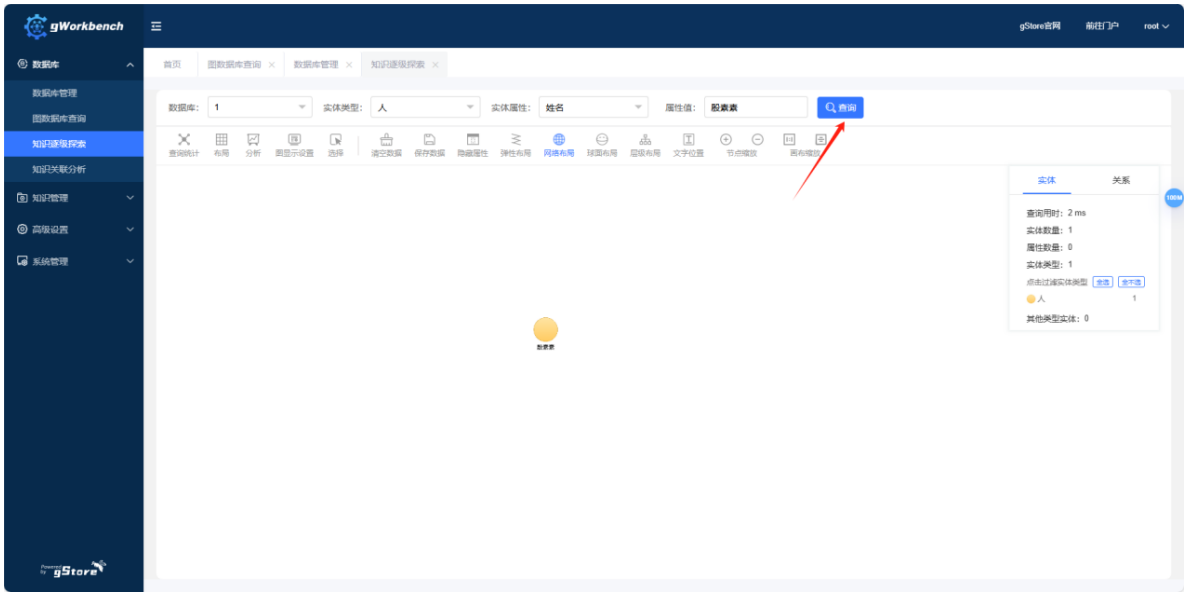


7.3.3 Knowledge hierarchical exploration

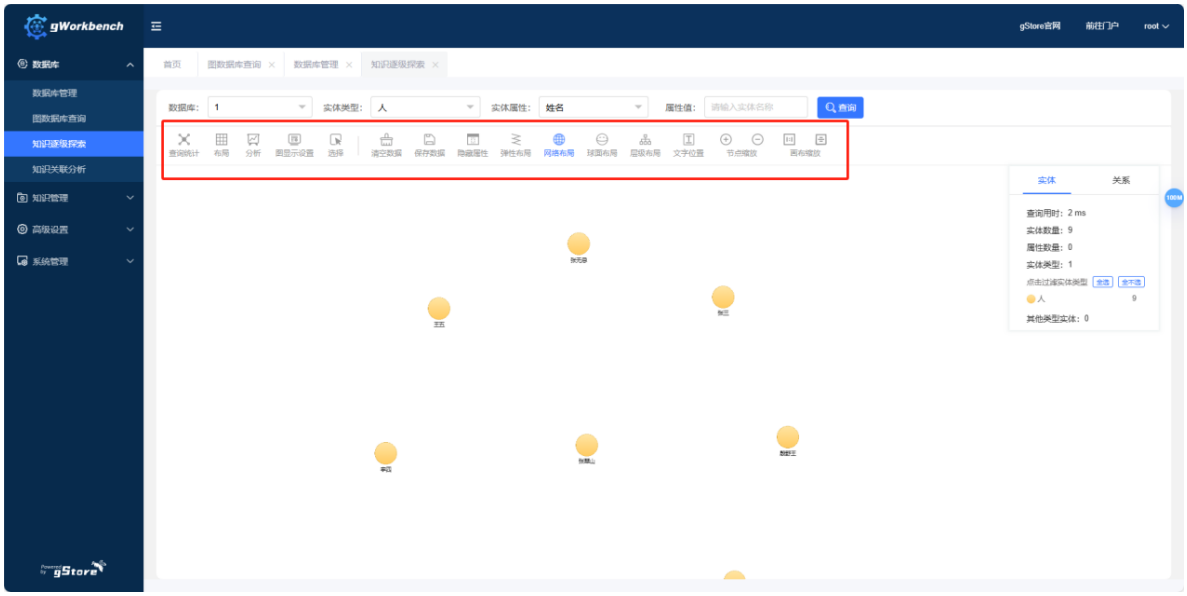
Knowledge hierarchical exploration allows users to query by selecting the database, entities, entity attributes, and attribute values, simplifying data retrieval. Except for attribute values, the other three fields are selected from dropdown menus, refining the query step by step from left to right.



After completing the information, the query can be performed as shown below (the last item is optional).



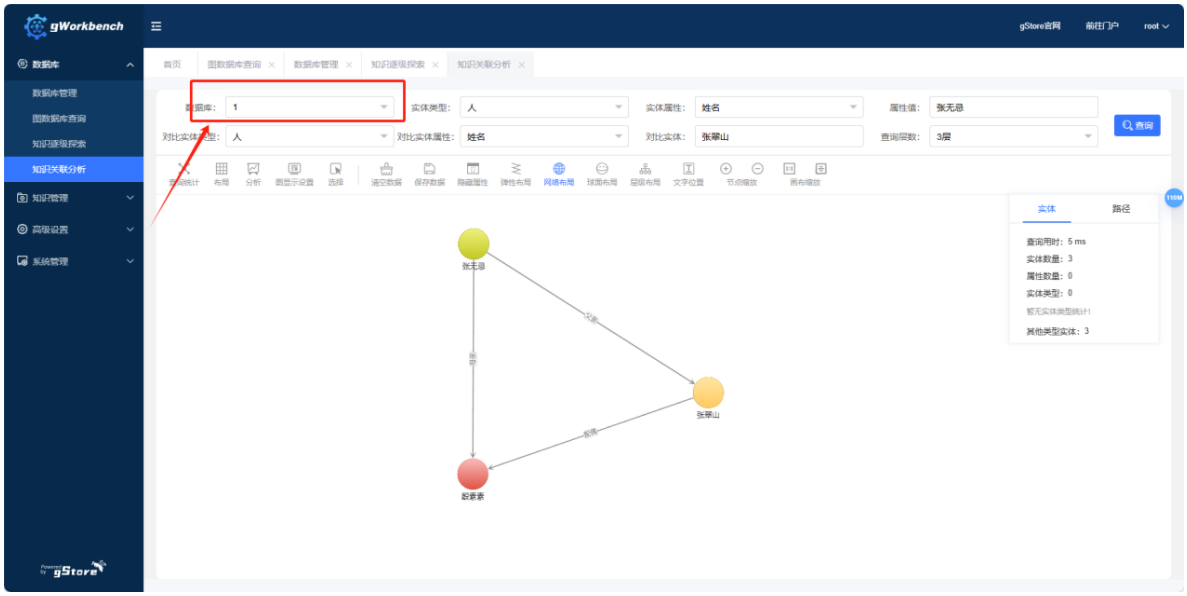
The functionality here is consistent with the one described in the basic query section, so it won't be elaborated further.



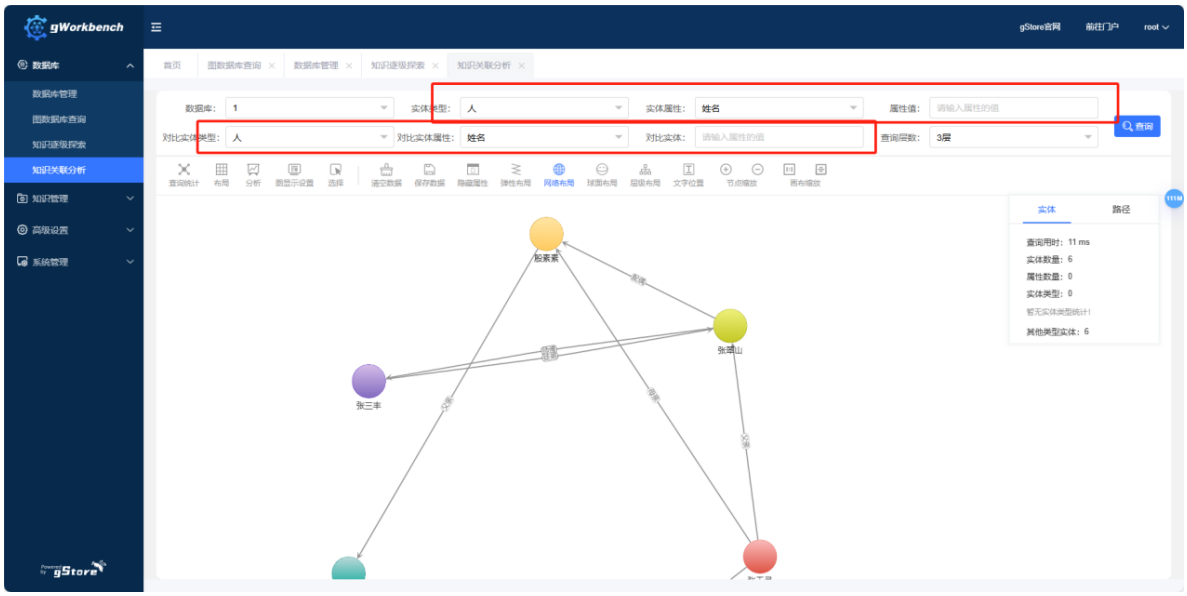
7.3.4 Knowledge association analysis

The Knowledge Association Analysis interface is designed to help you explore the complex relationships between entities, entity types, and entity attributes. Through knowledge association analysis, you can compare relationships between different entities and query the associations between two entities within a specified number of hops, revealing hidden deep relationships in the knowledge graph. For clarity, we'll refer to the two nodes being analyzed as A and B, focusing on the relationship between them.

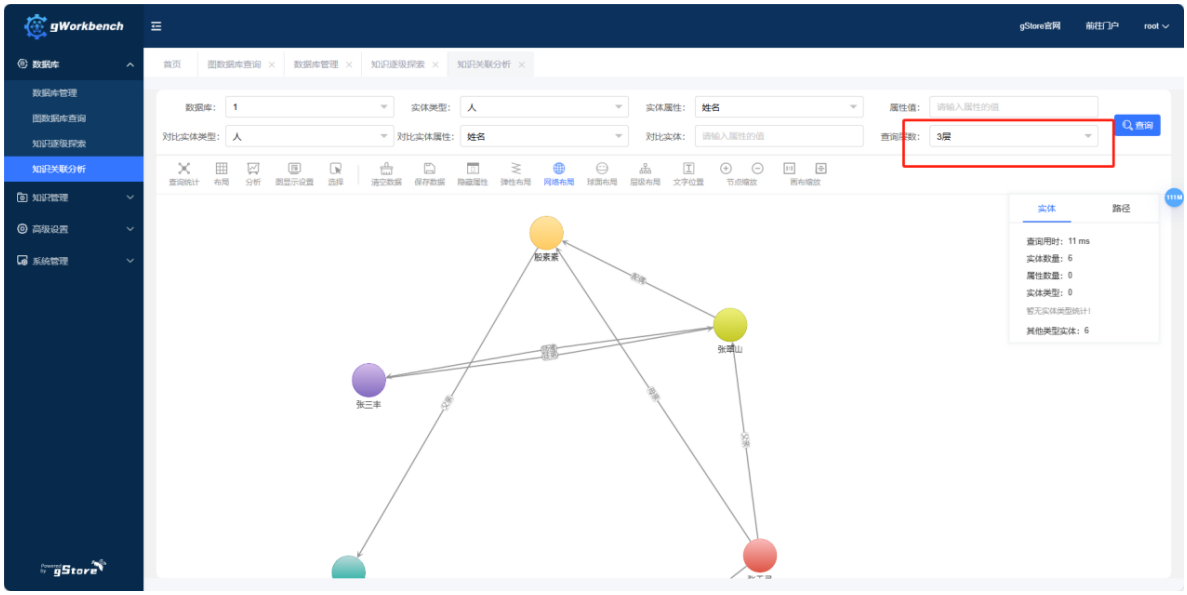
Select the database to use:



The two nodes selected within the red boxes represent the nodes we've chosen (note that if no attribute values are entered, the relationship between the set of entities with that attribute and another node is analyzed. If both nodes are selected with attributes, you can consider it as a relationship between two sets, each containing only one element).



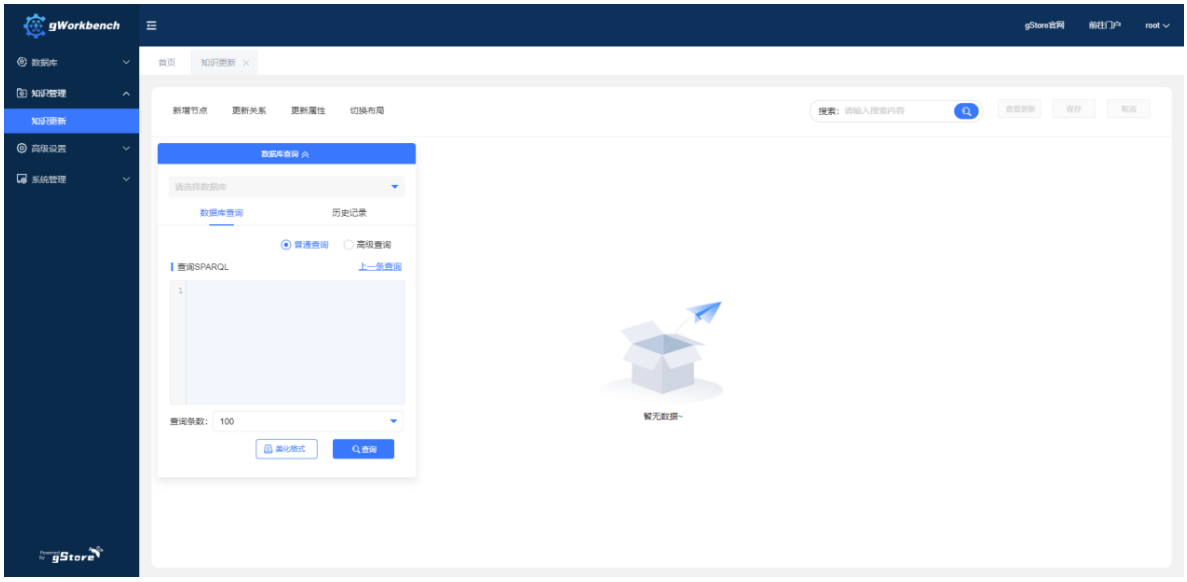
The number of hops indicates the maximum number of steps needed to reach the target node:



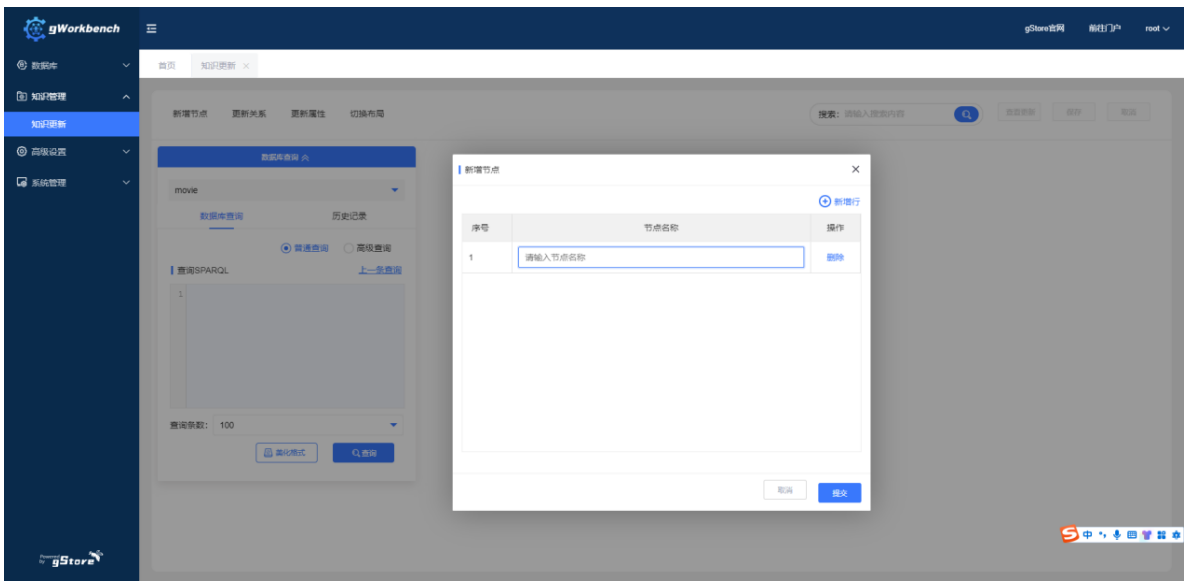
7.4 Knowledge Management

7.4.1 Knowledge Update

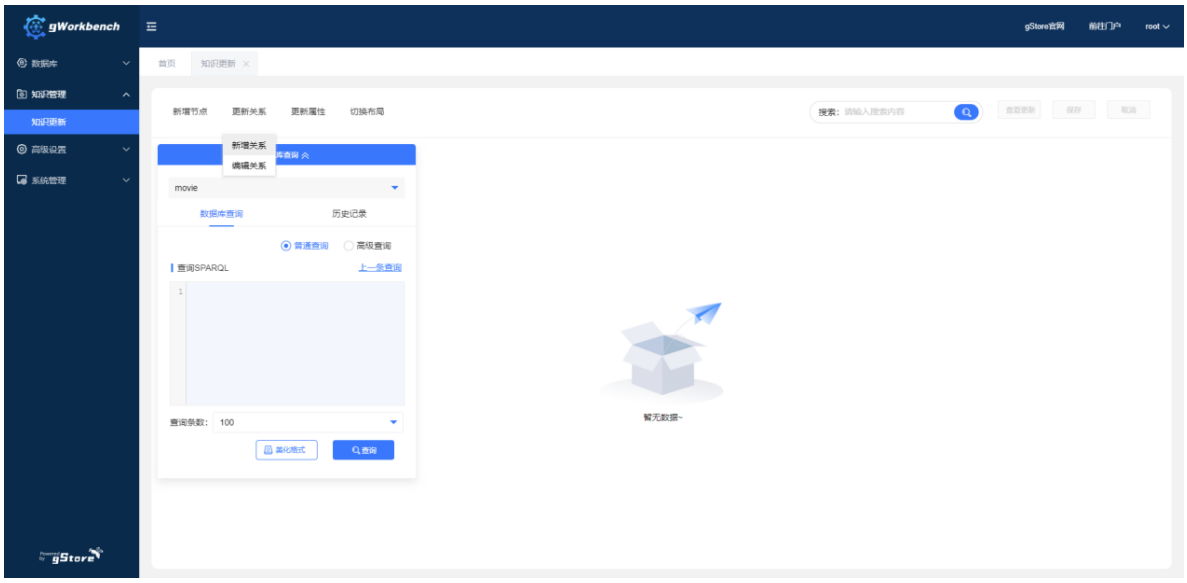
Knowledge update includes adding new nodes, updating relationships, updating attributes, and switching layouts. It also allows for database queries and viewing of historical records.

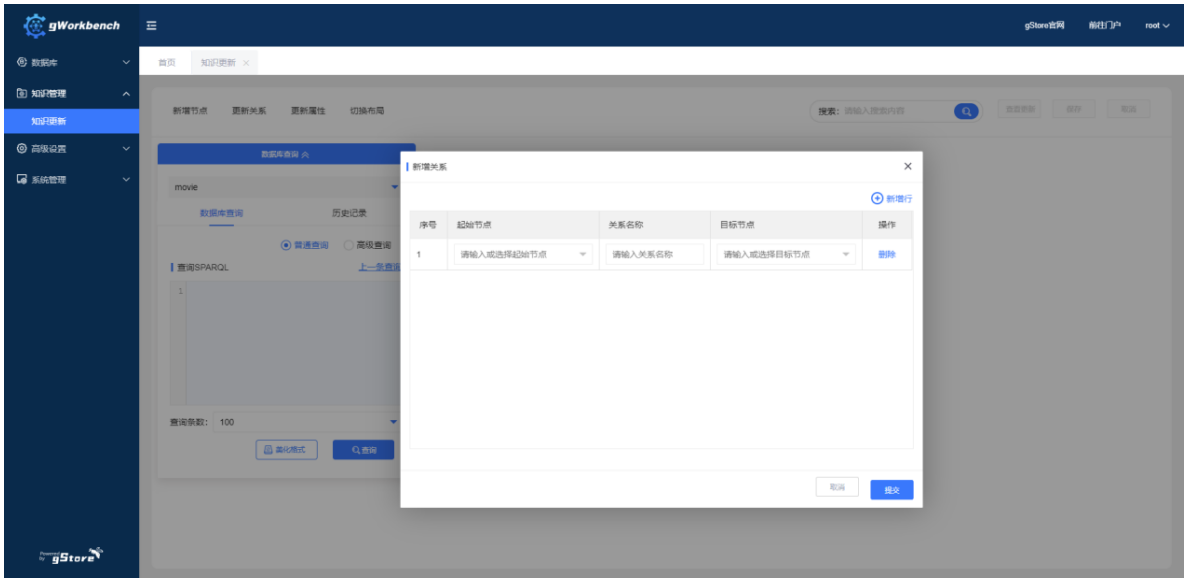


Click 'Add New Node' to add multiple node names:

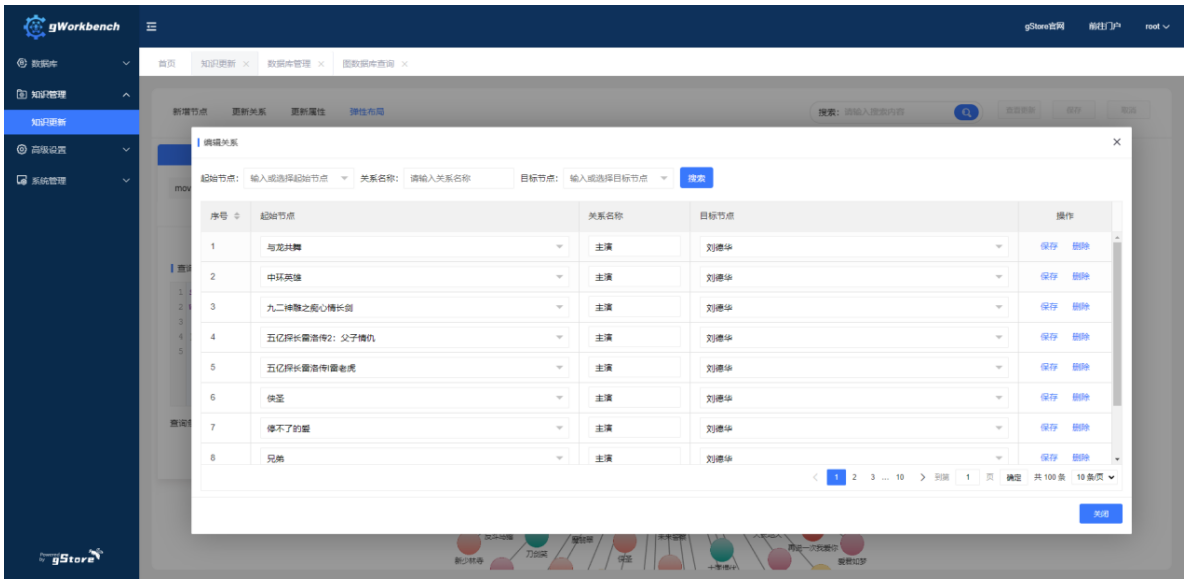


Click 'Update Relationships' to select 'Add Relationship' or 'Edit Relationship'. Click 'Add Relationship' to add multiple node relationships:

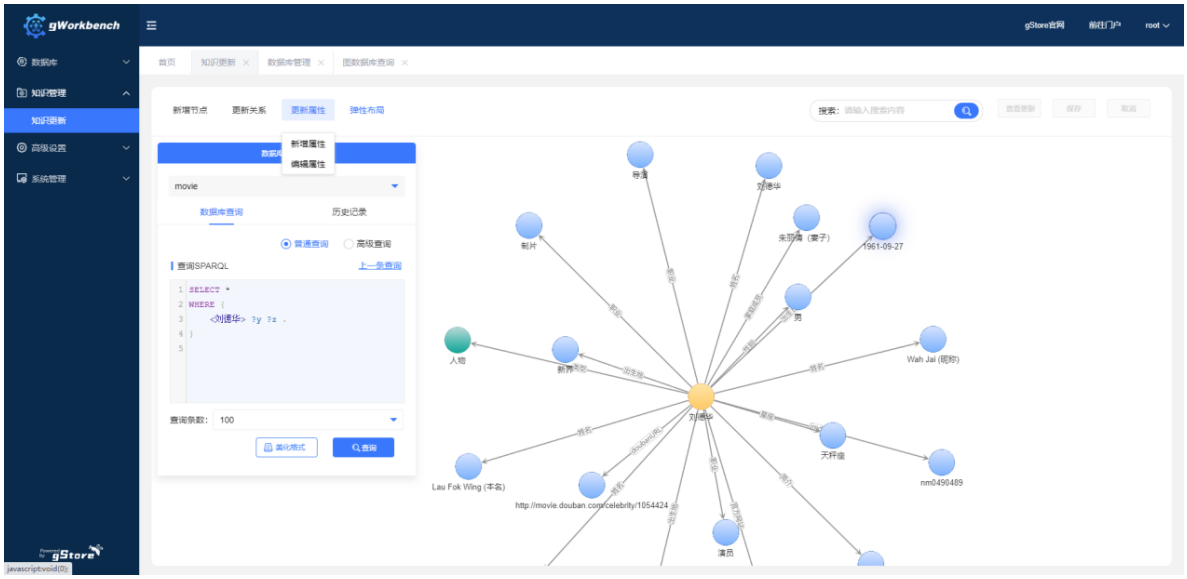


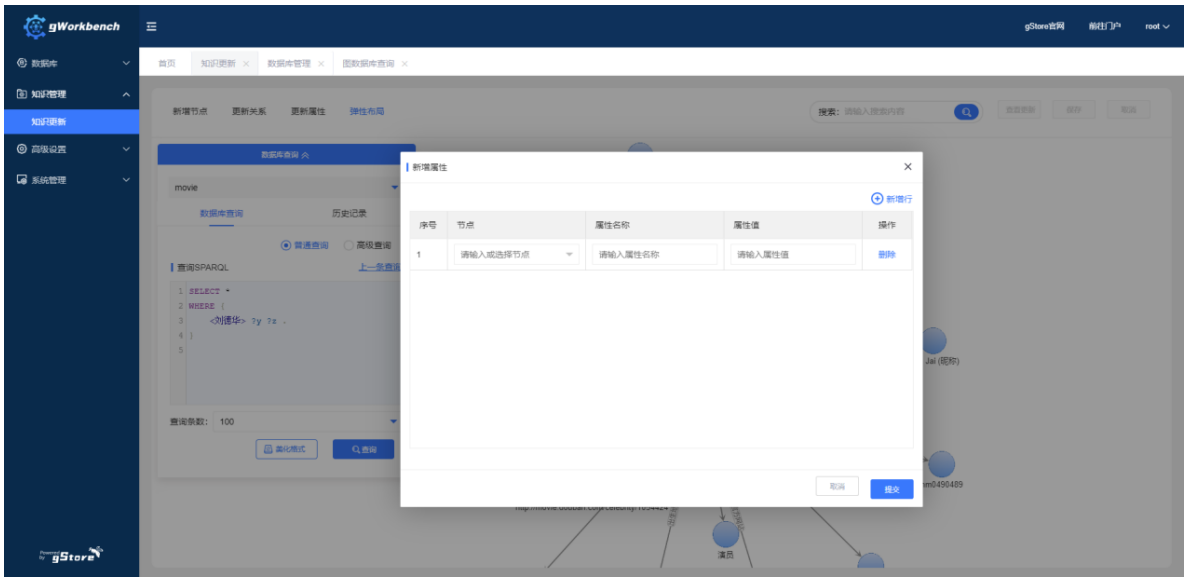


In 'Edit Relationships,' you can edit the nodes, relationships, and node information in the queried database:

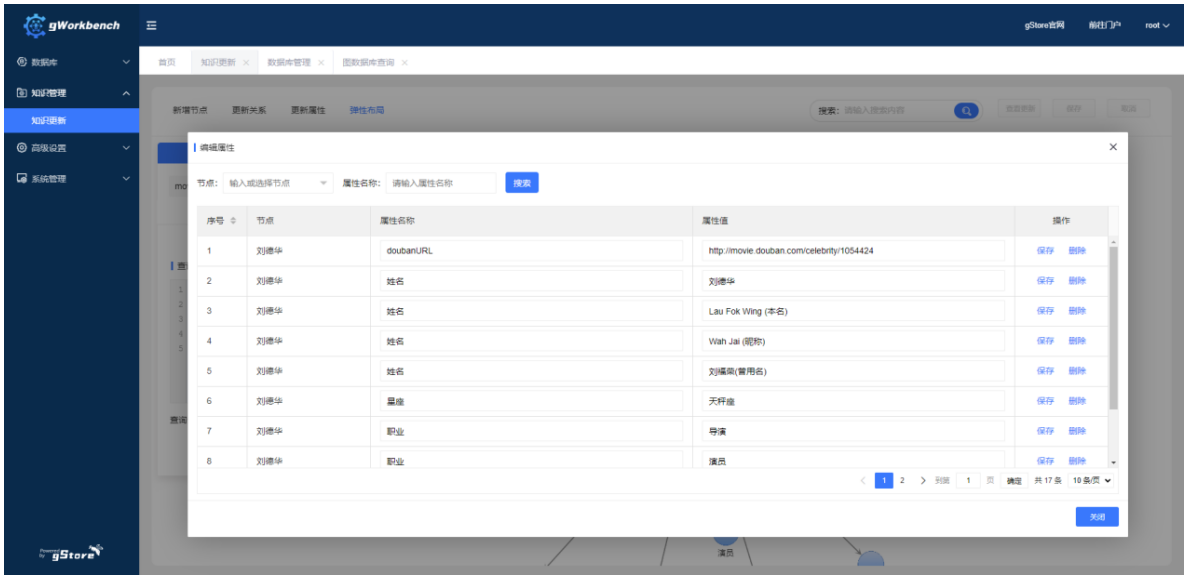


Click 'Update Attributes' to select 'Add Attributes' or 'Edit Attributes.' Click 'Add Attributes' to add multiple nodes, attribute names, and attribute values:

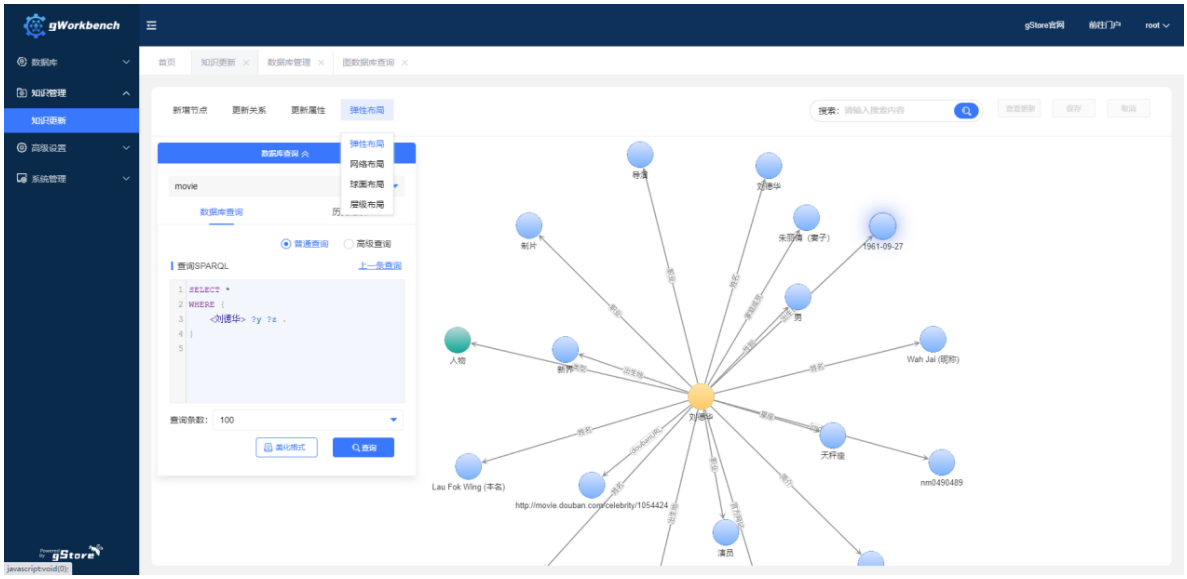




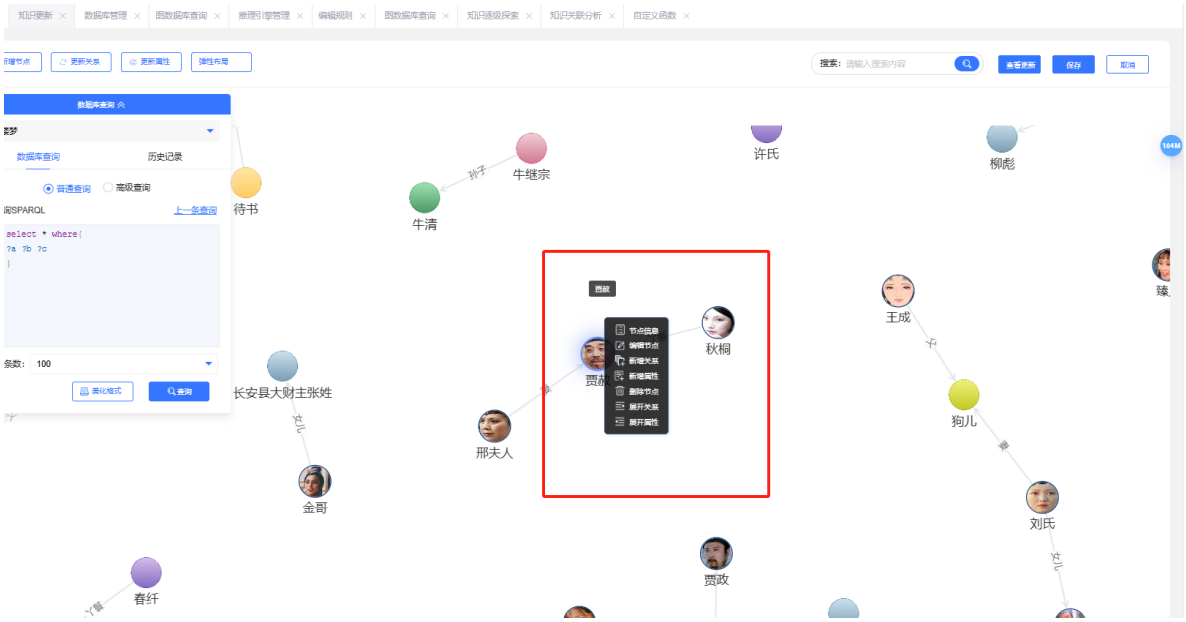
Click 'Update Attributes' to edit attributes, allowing you to query, modify, and delete nodes, attribute names, and attribute values:



The elastic layout supports various graph layout forms:

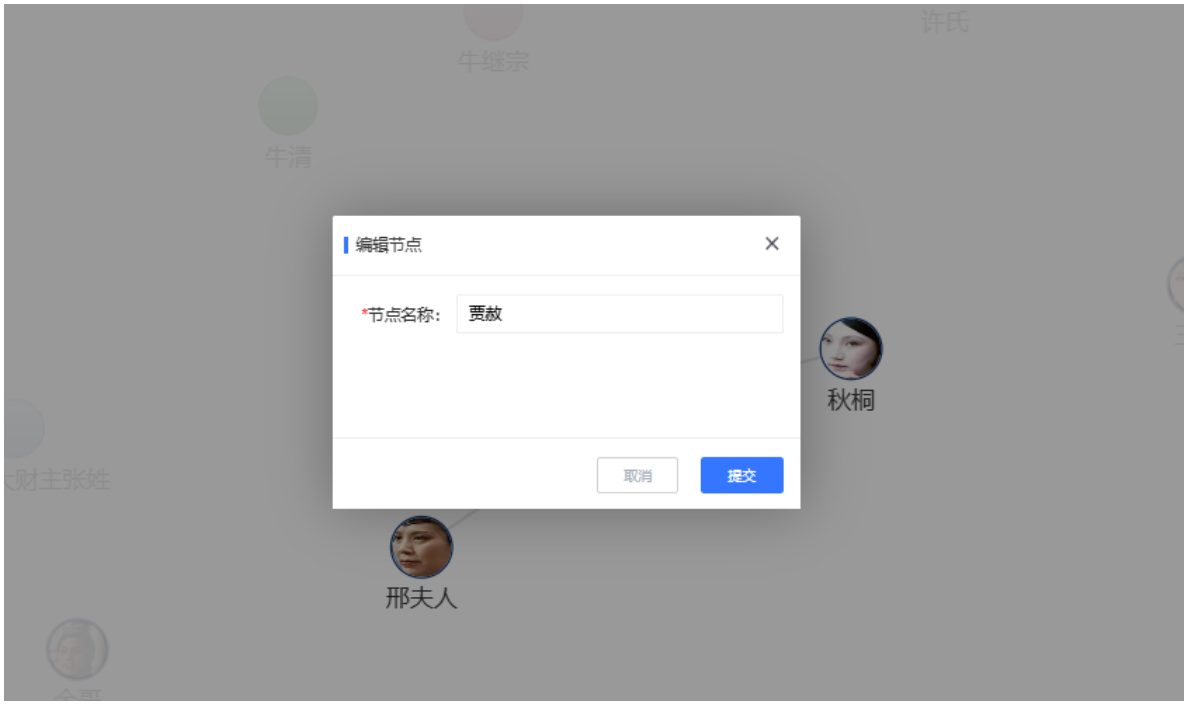


Right-click on an entity node to access the following functions:

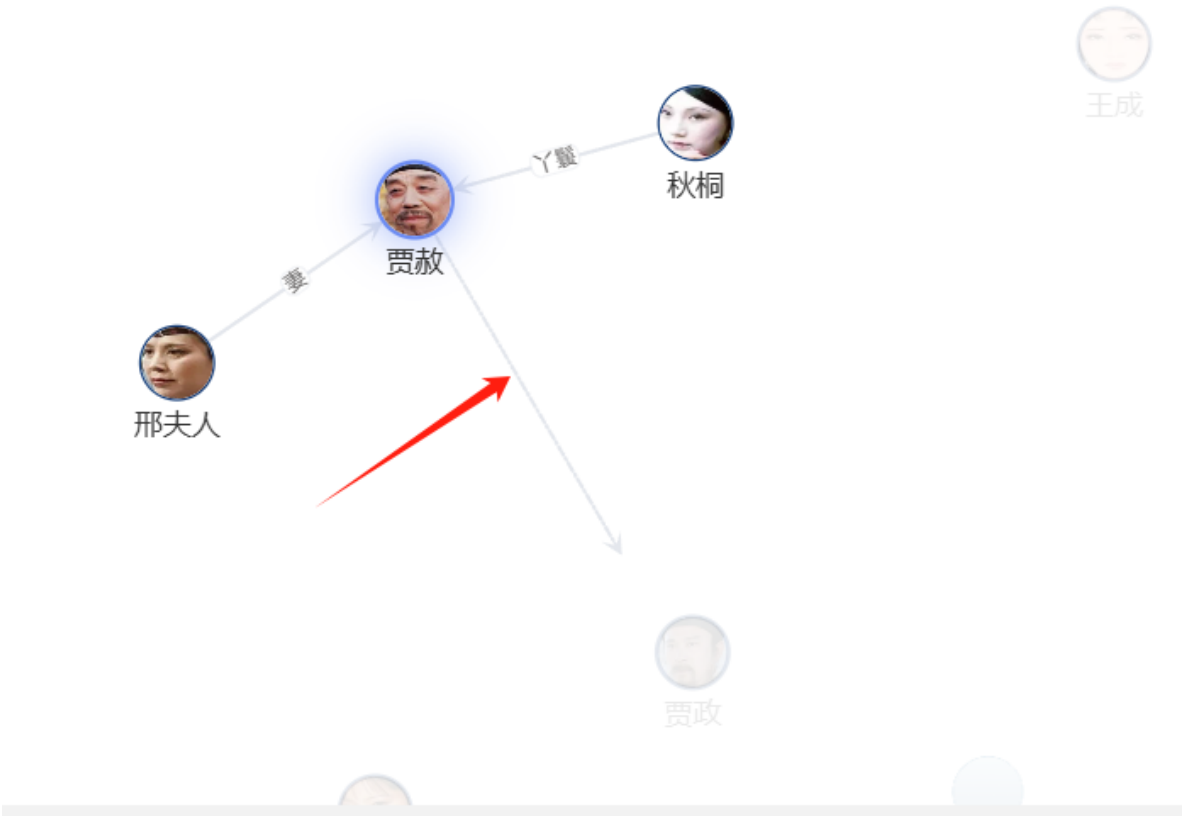


(1) Node Information:

(2) Edit Node (Modify Node Name):

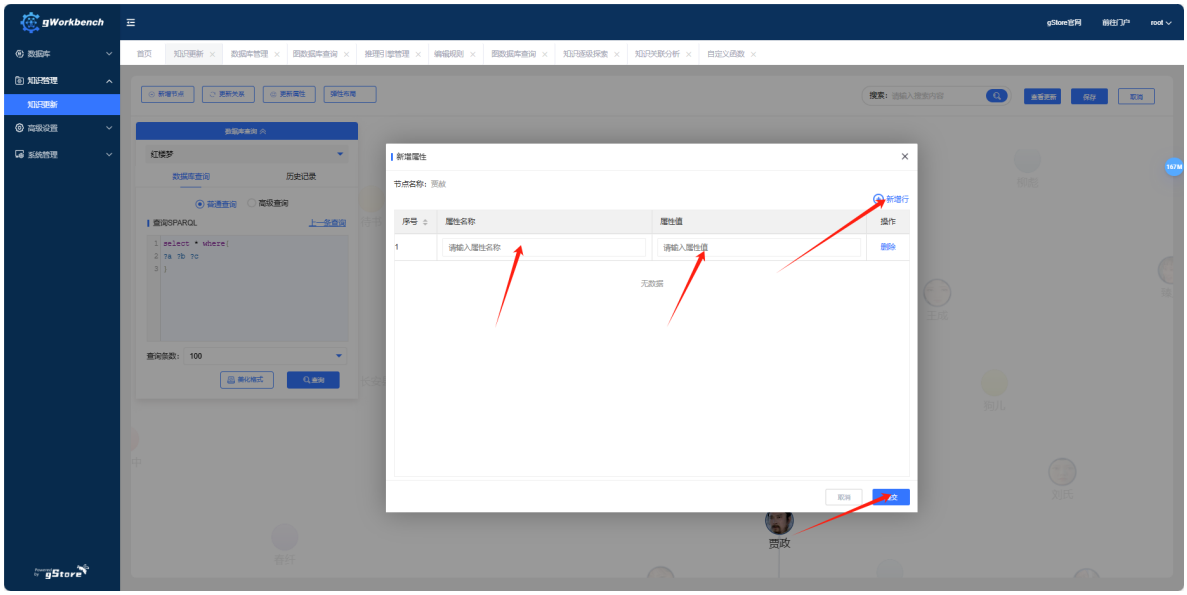


(3) Add Relationship (Draw a line pointing to another entity node):

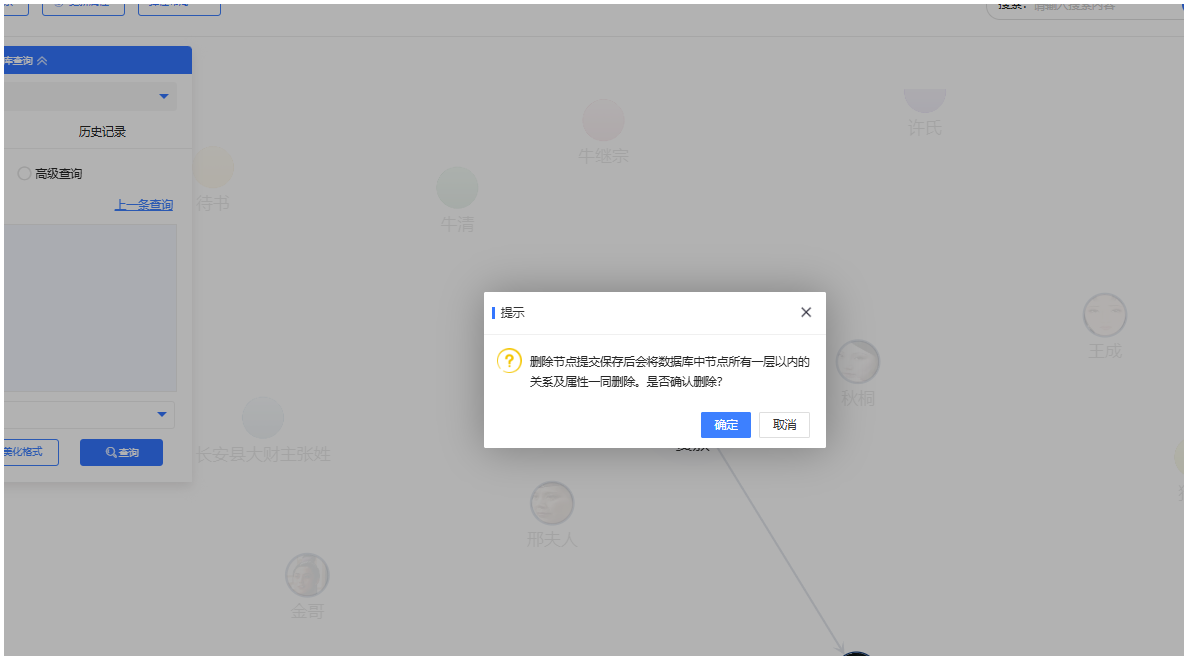


Enter the relationship name and submit to save.

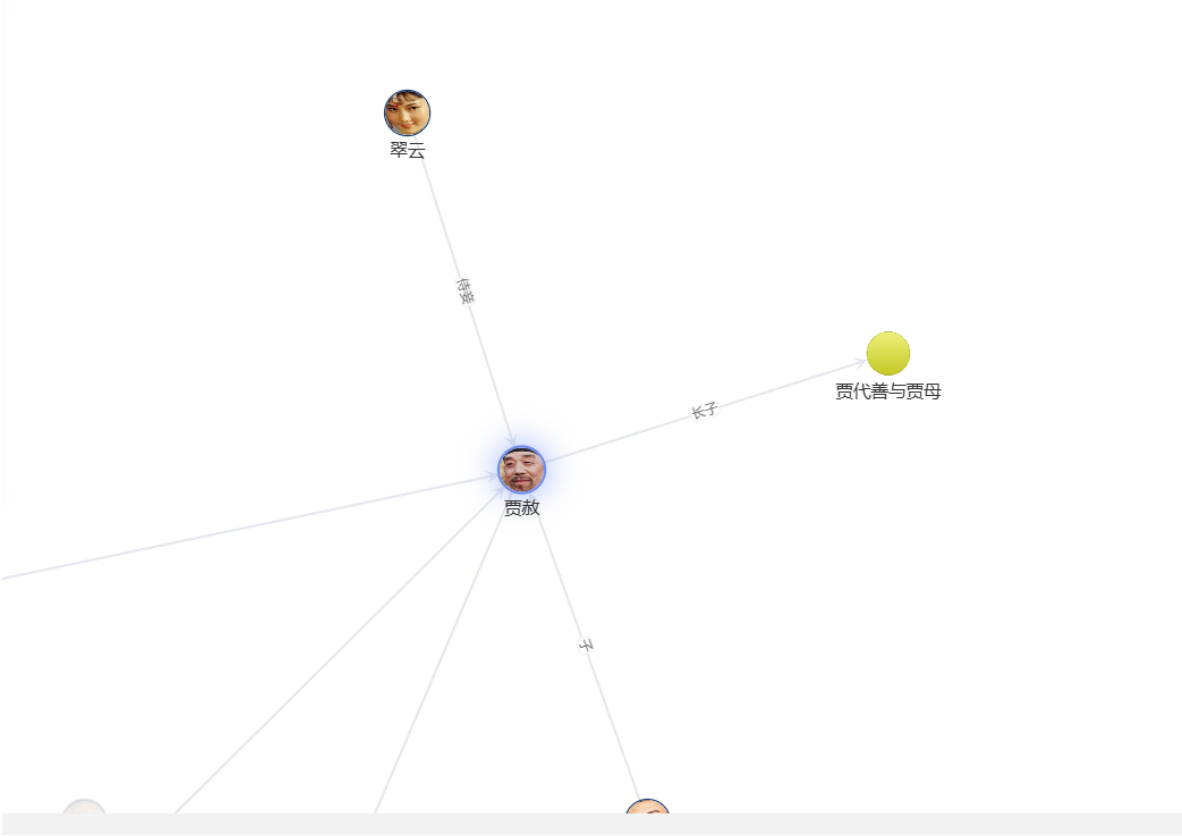
(4) Add Attribute (Same as the 'Add Attributes' operation described above):



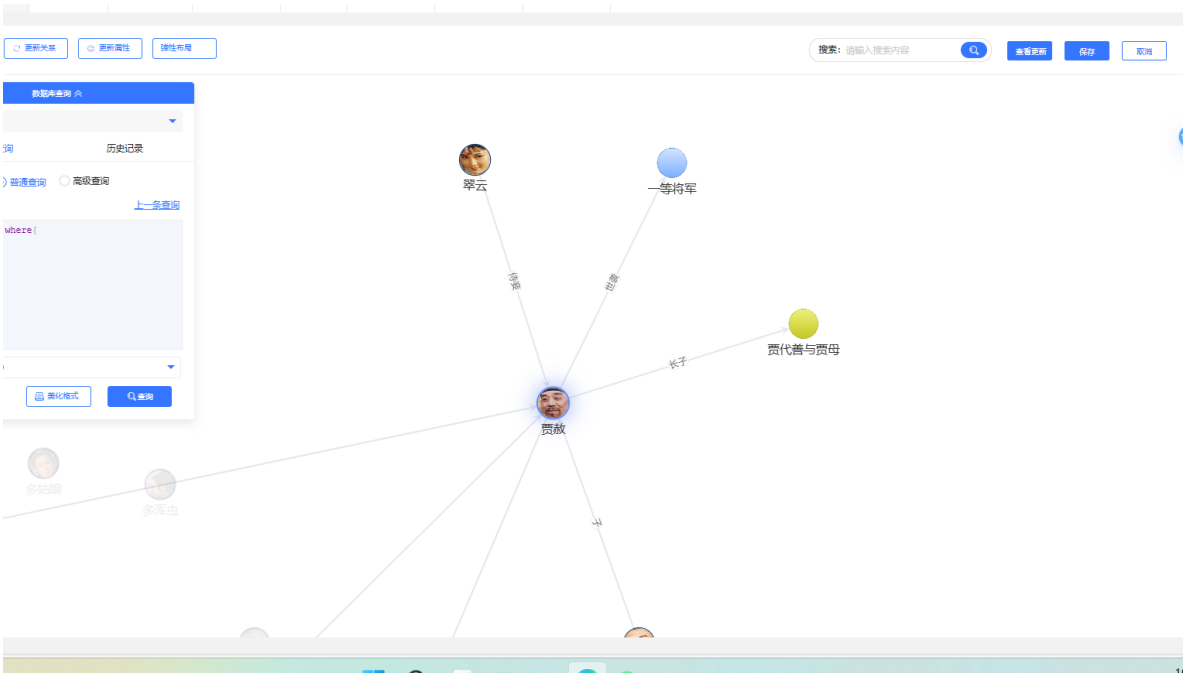
(5) Delete Node (Remove the entity node)



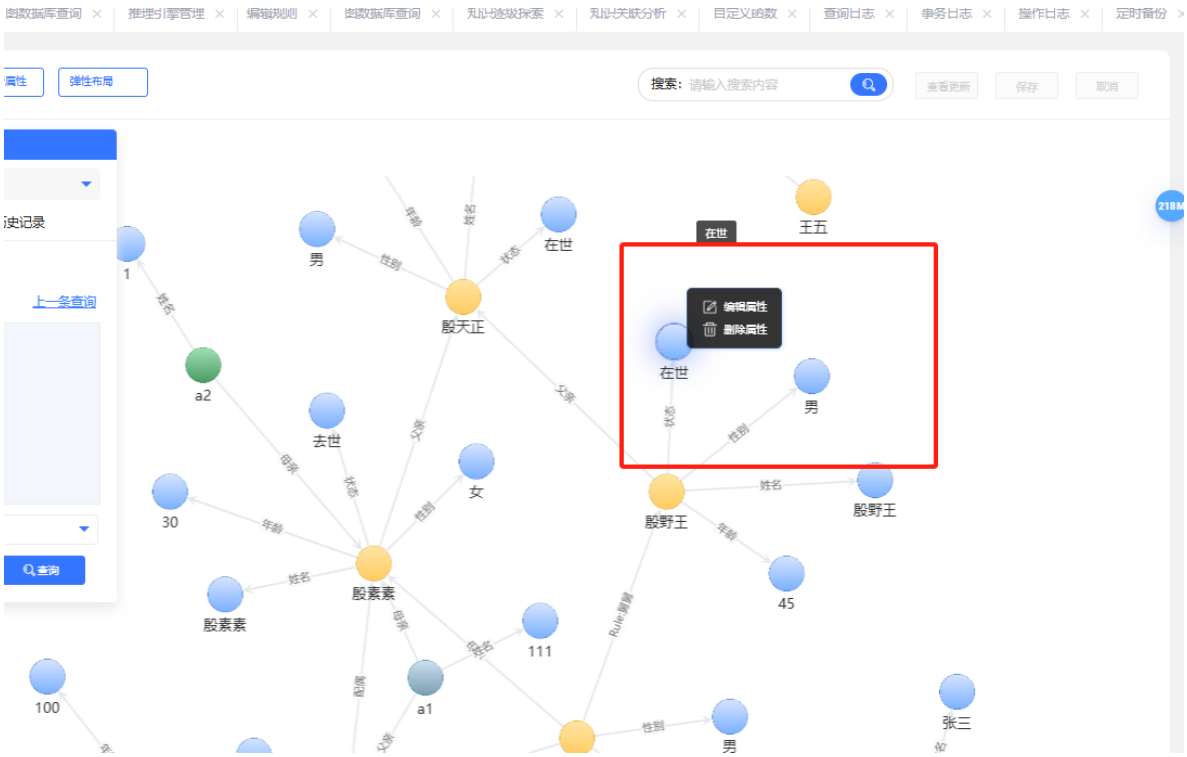
(6) Expand Relationships (Display nodes related to the selected node):



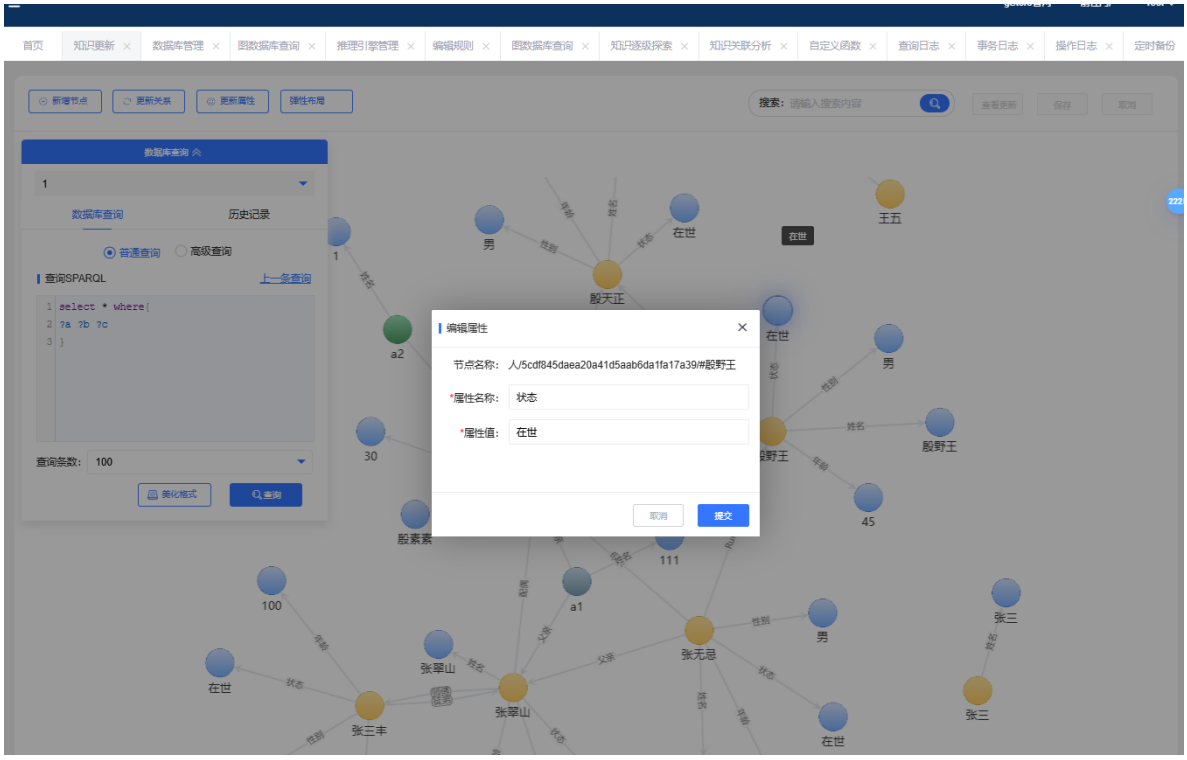
(7) Expand Attributes (Show the attributes of the selected node):



Right-click on an entity node's attribute to access the following functions:

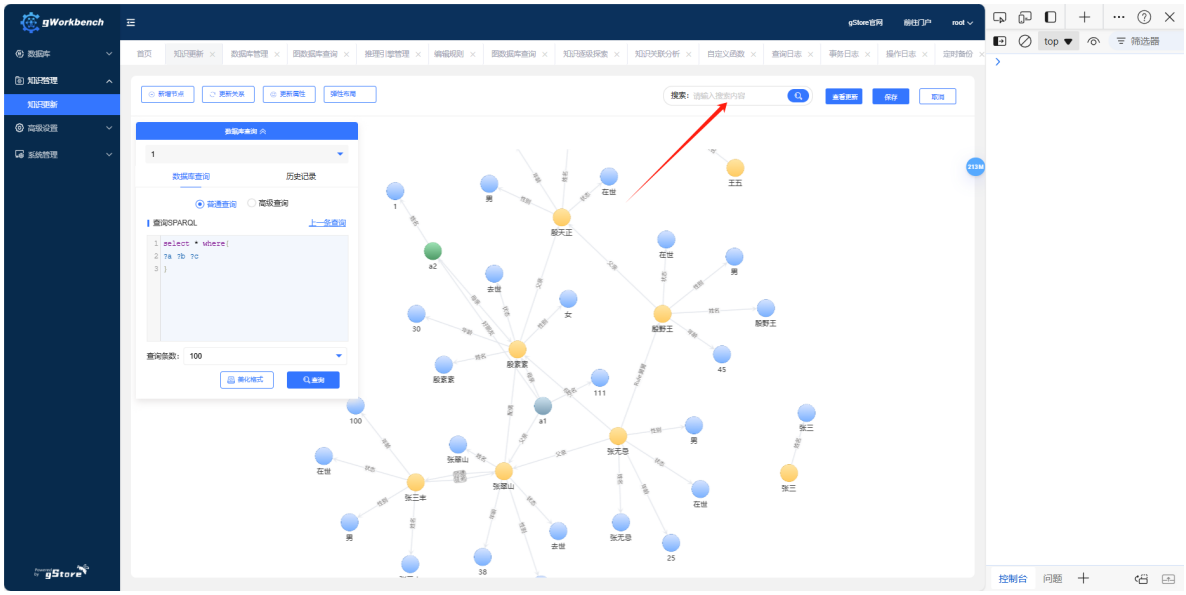


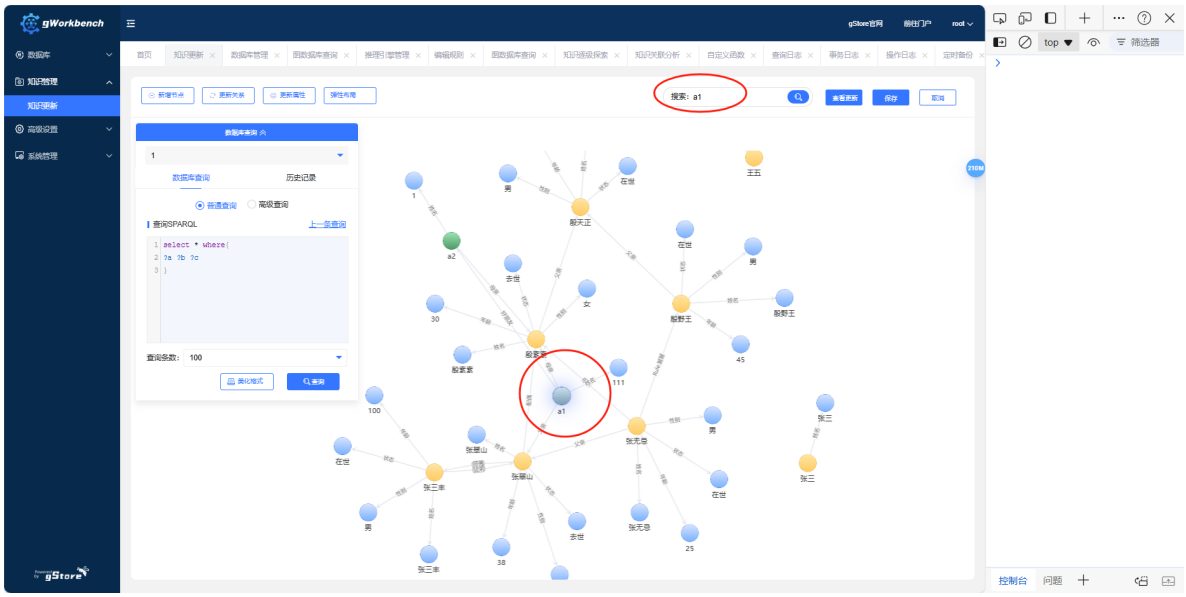
(1) Edit Attribute (Change the attribute name and value):



(2) Delete Attribute (Remove the attribute from the node):

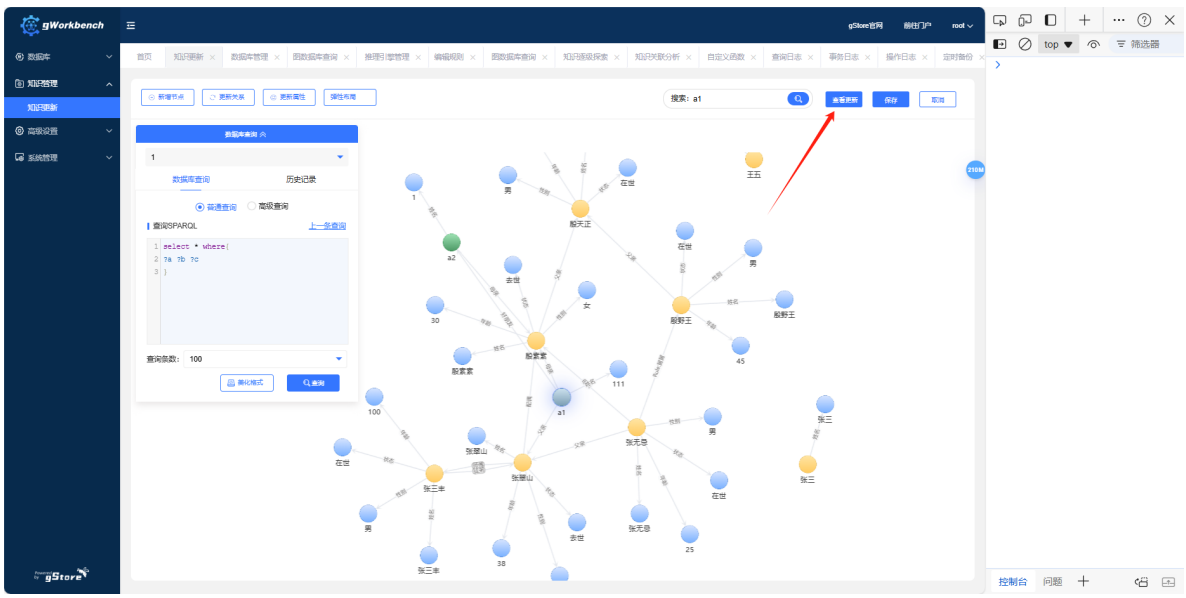
Search (Enter a node name to find its location on the graph):



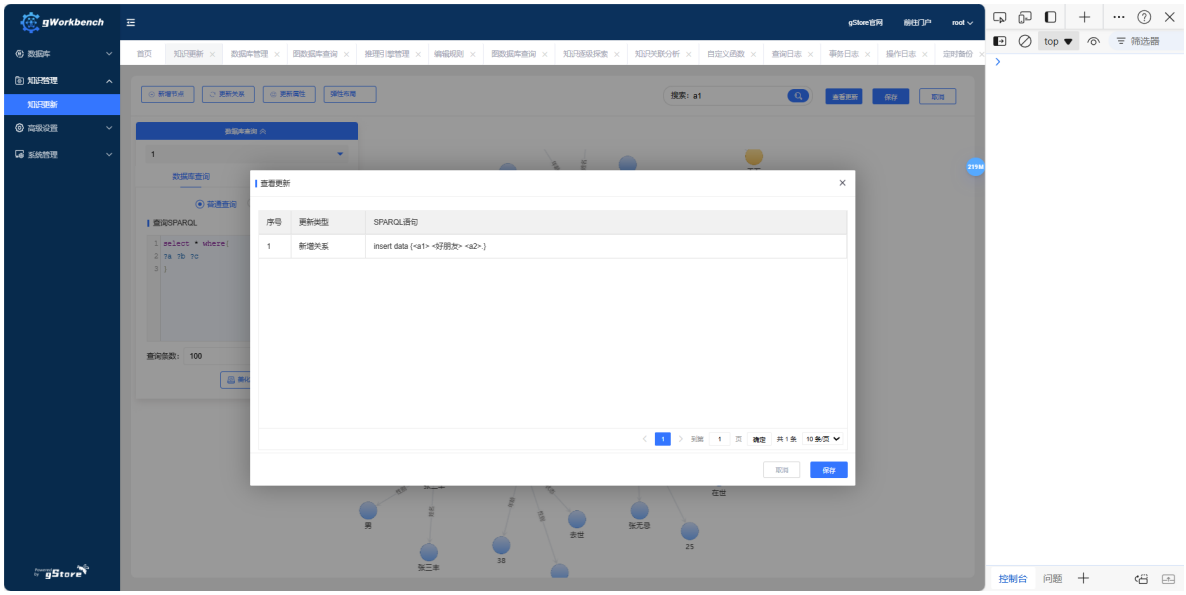


Check Updates:

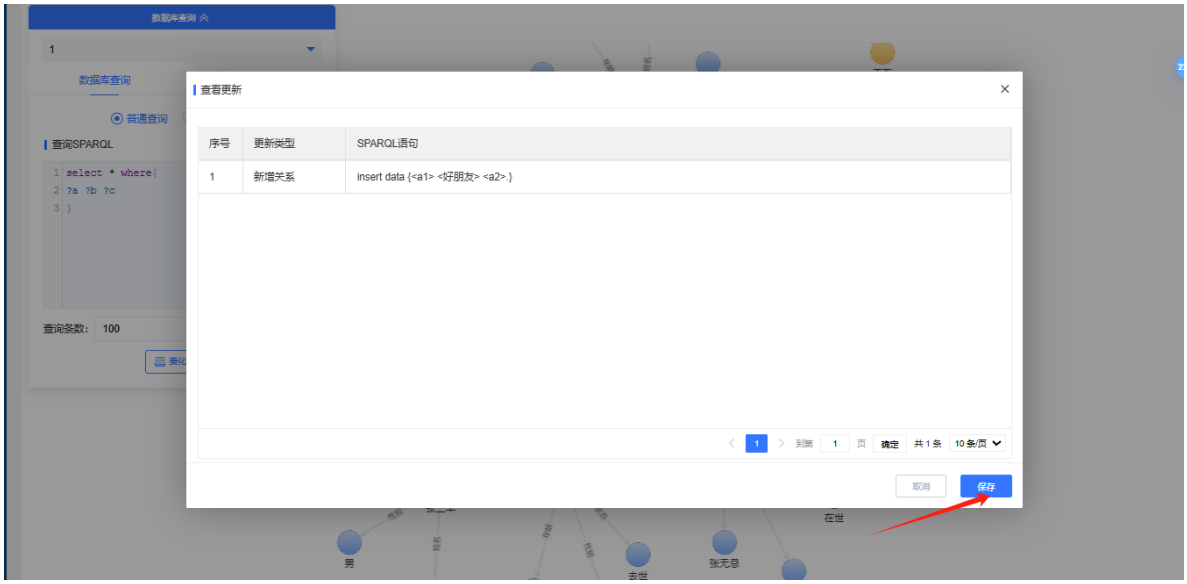
(1) Where the action buttons are located:



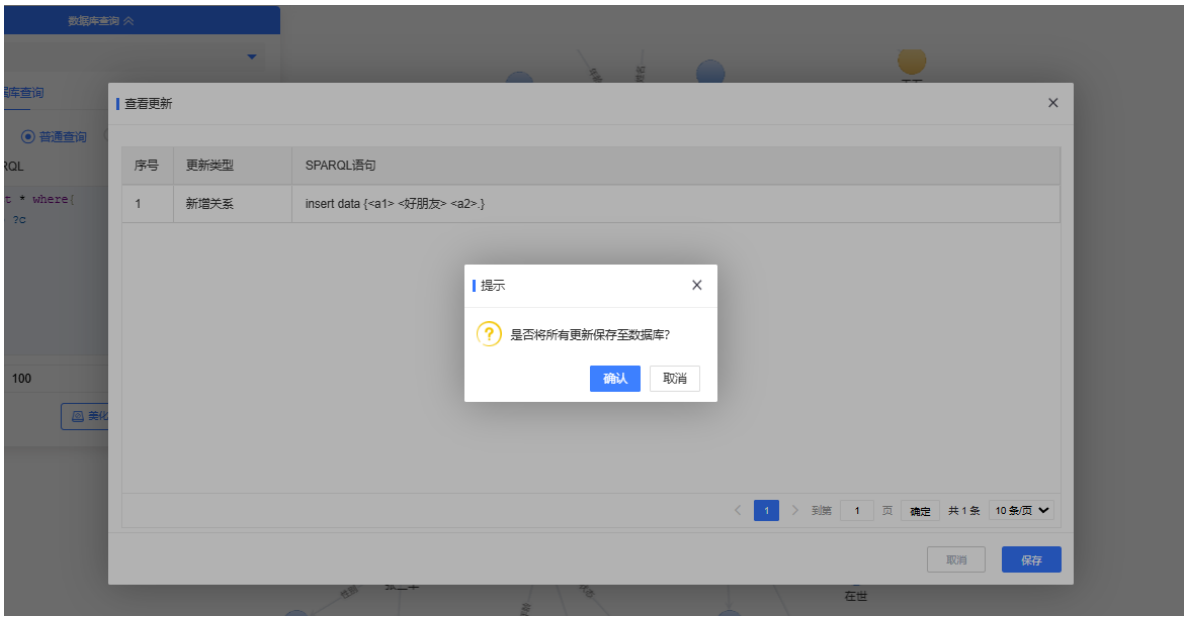
(2) Displayed content:

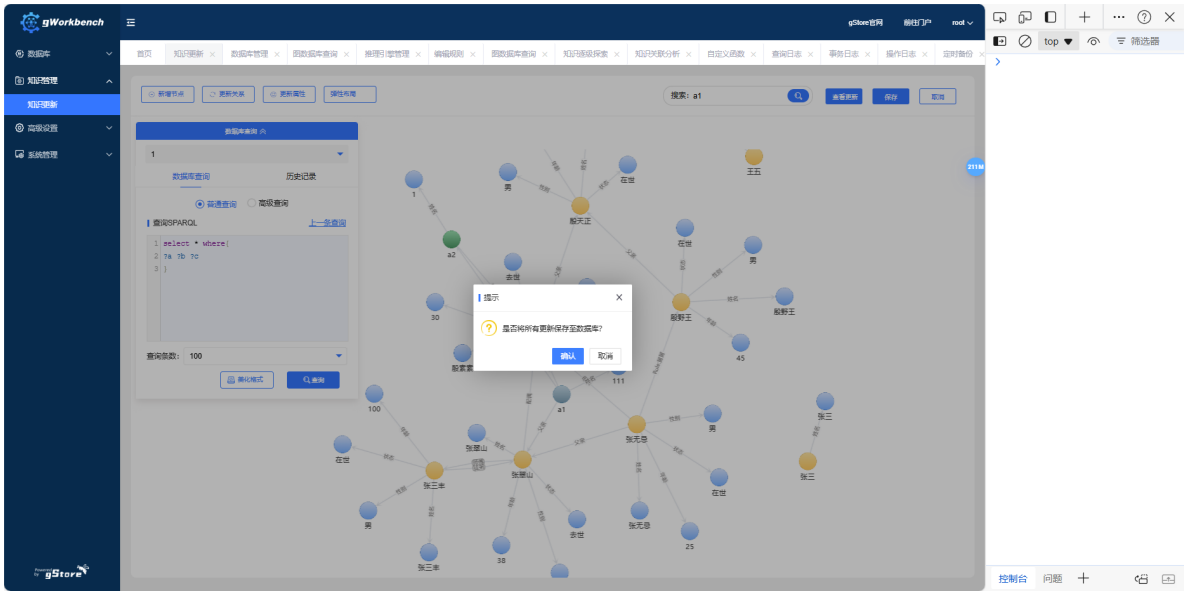


(3) Save:

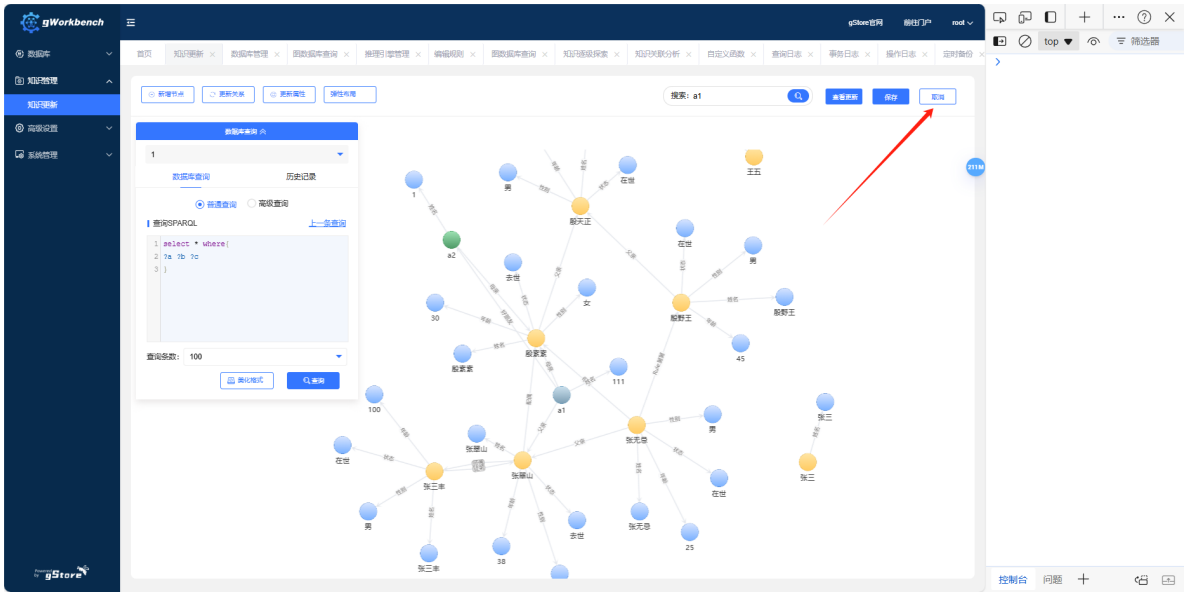


Save Knowledge Updates:





Cancel Knowledge Updates (Revert unsaved update operations):



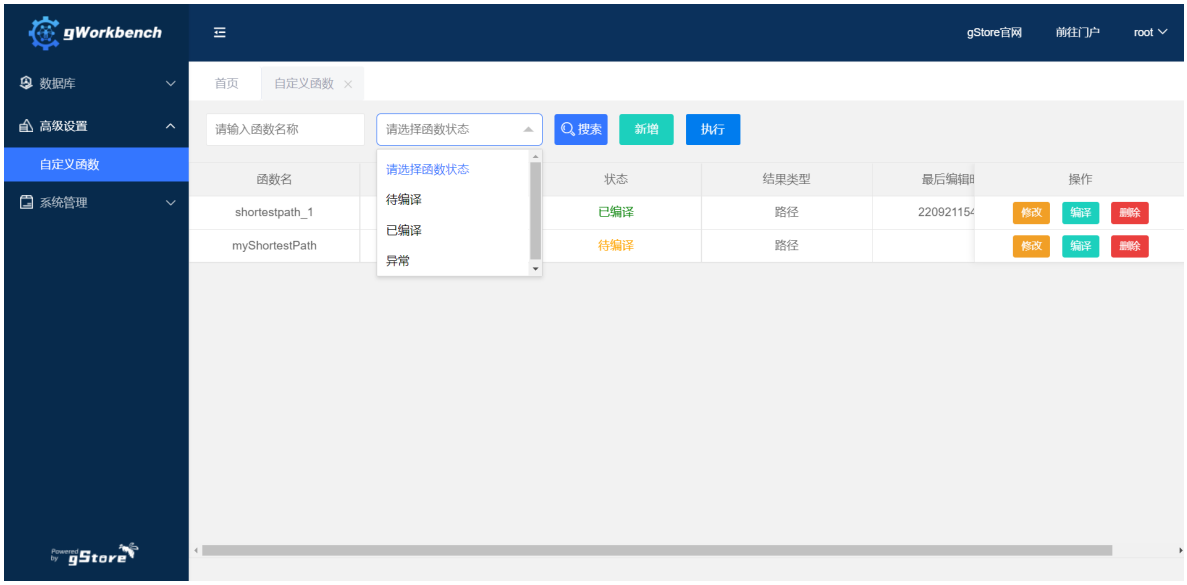
7.5 Advanced settings

Users can define custom graph analysis functions through the advanced settings module, which, after definition, can also be called directly from the graph database query module.

7.5.1 Custom function

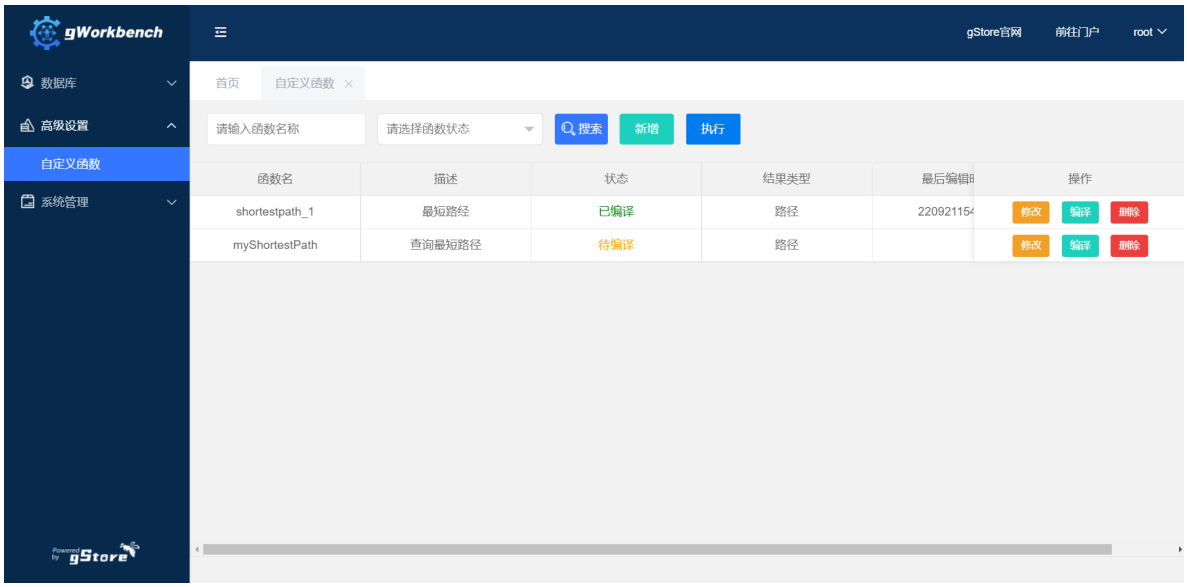
Find existing custom function

Click the [Advanced Settings] - [Customized Function] module, enter the name of the custom function to be queried, select the function status, and click [Search] to find the target custom function.

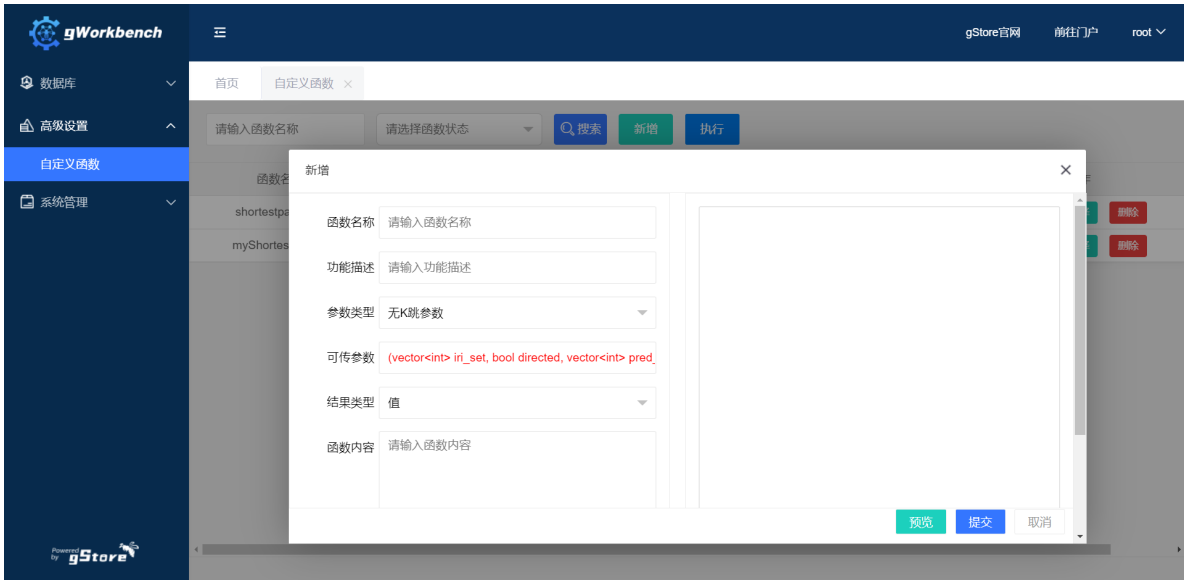


Create new custom function

Click the [Advanced Settings] - [Custom Function] module, and then click [New].

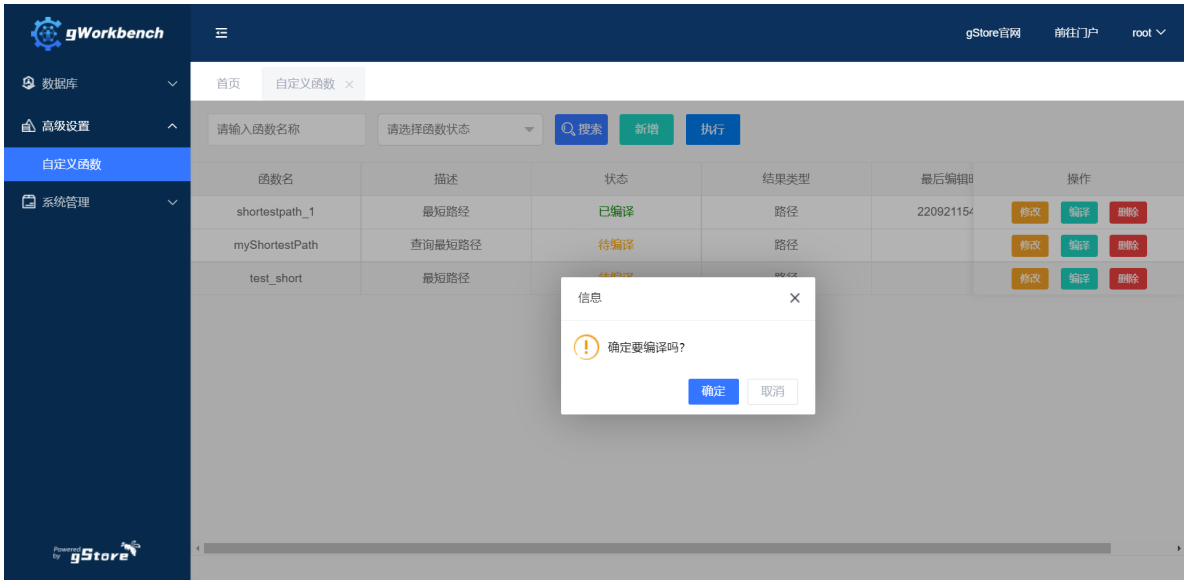


Enter the function name, function description, and parameter type.

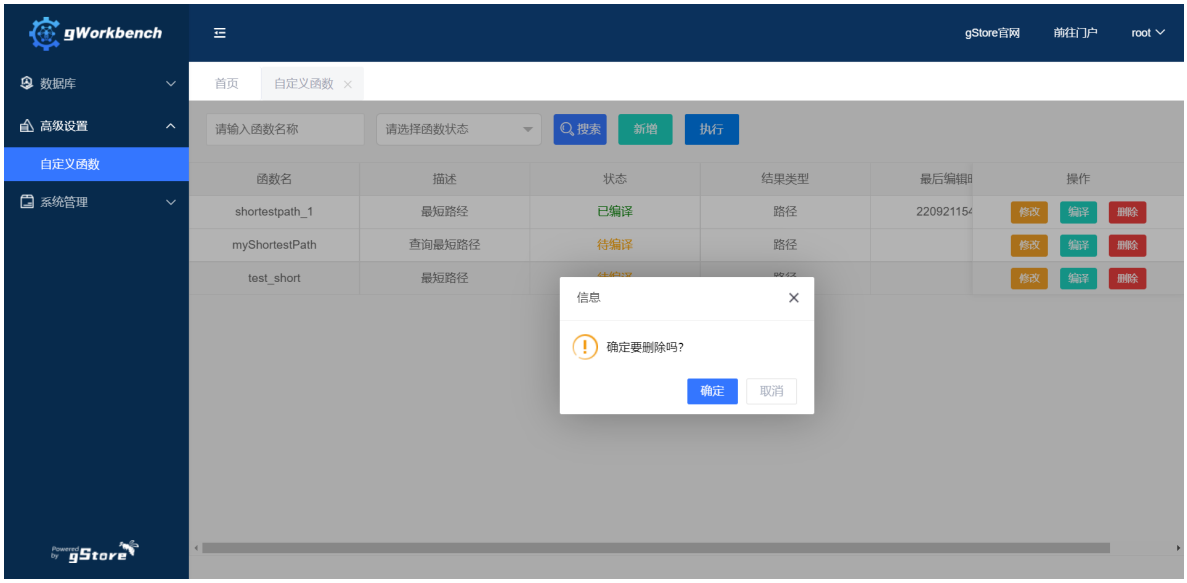


Click [Preview] to view the whole function.

Click [Compile] to compile the custom function.



Users can also click [Delete] to delete custom functions.

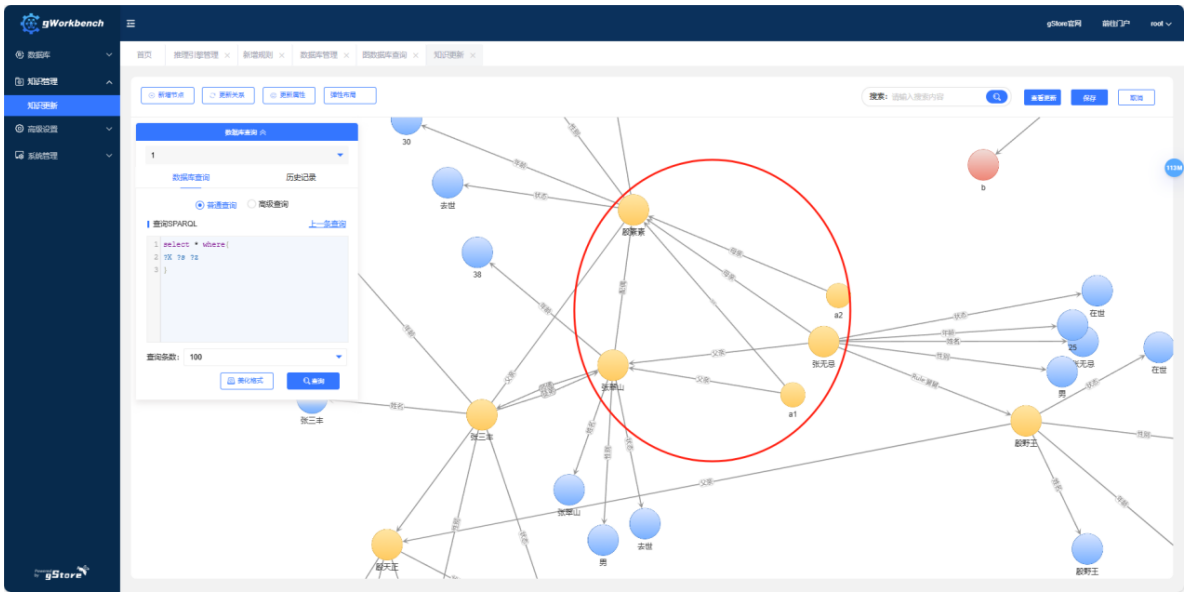


Execute custom function

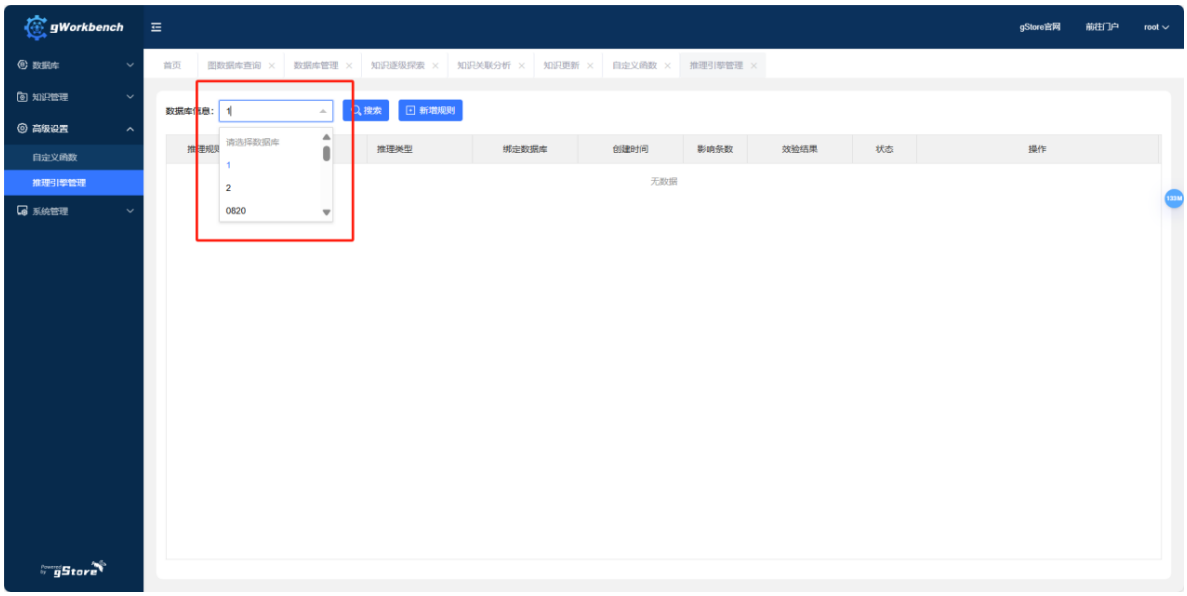
Click [Advanced Settings] - [Custom Function] module, click [Execute], input the database, function to be executed, source and/or destination nodes, K-hop value and other information, then click [Execute] to get the result.

7.5.2 Inference engine management

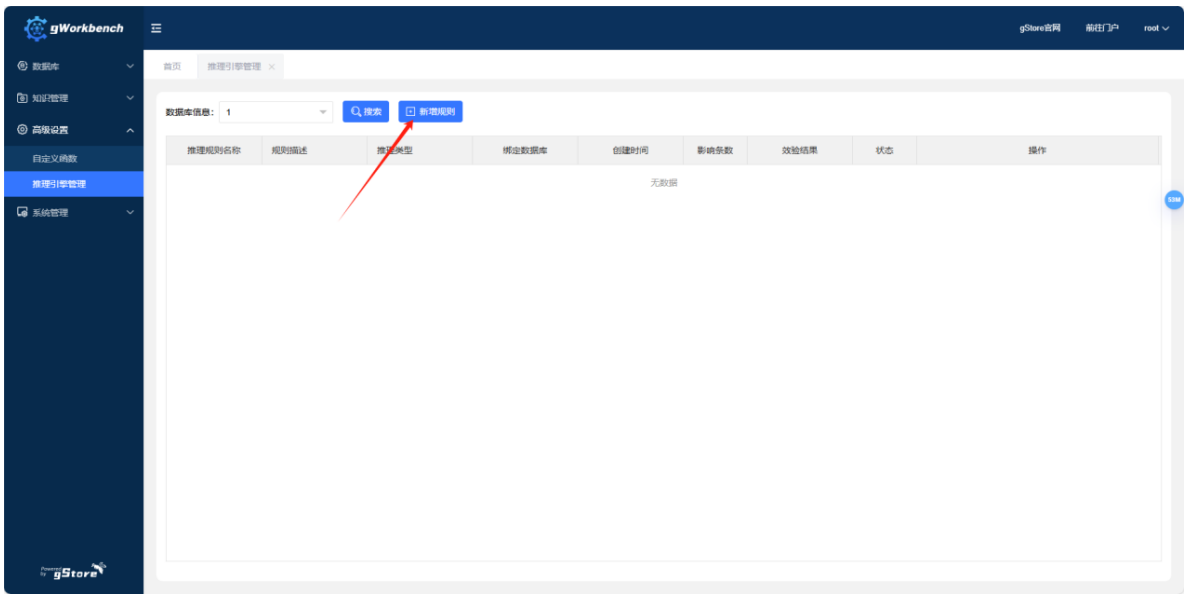
The inference engine automatically updates the database based on the information we input into the rules. I will provide an example below. I have prepared the following data:



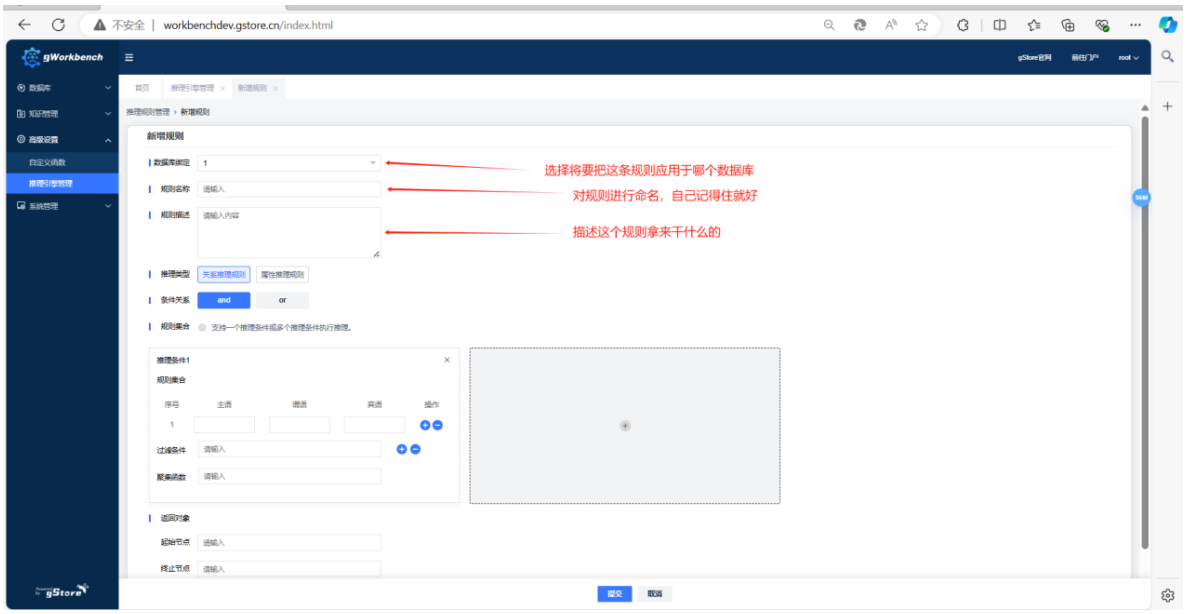
Select a database:



Add a new rule:

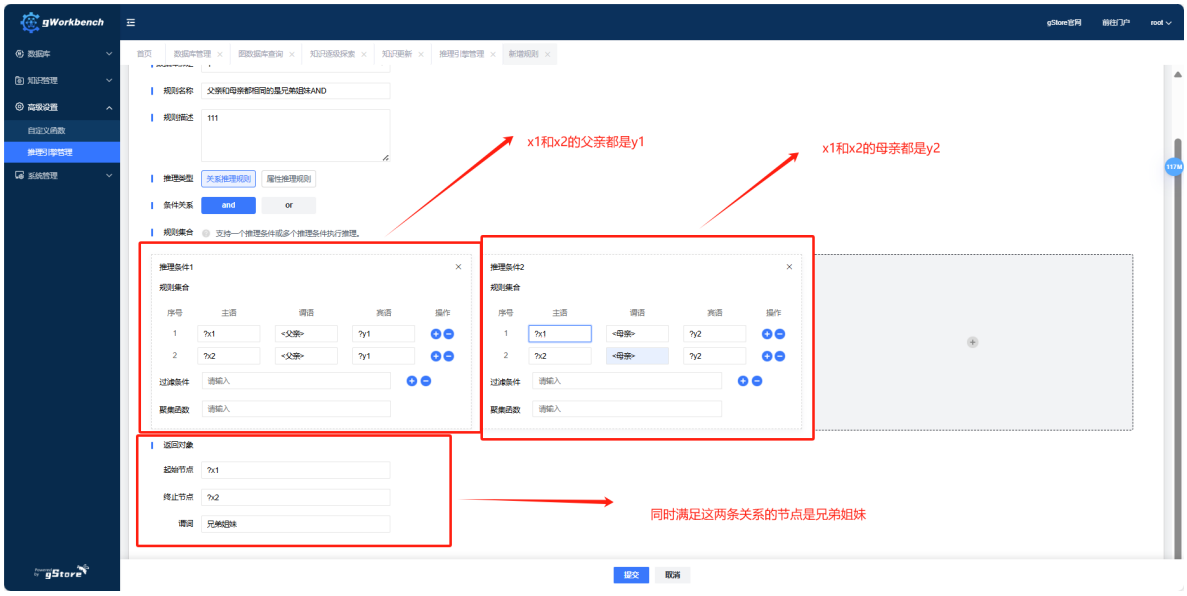


The functions of the first three options are as follows:

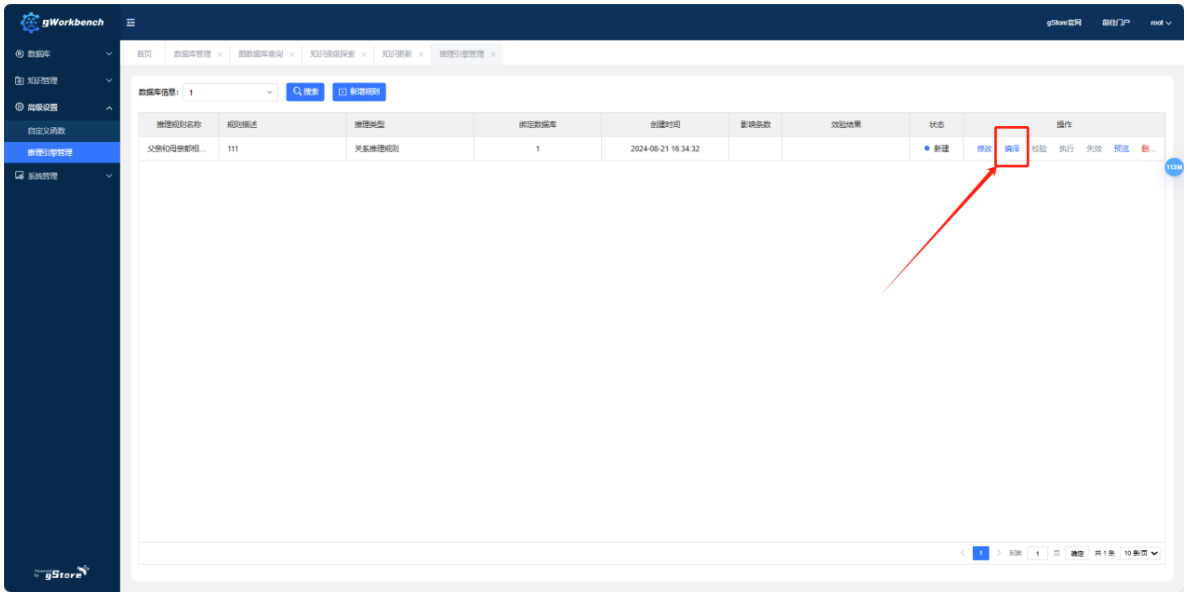


Relationship Inference Rules

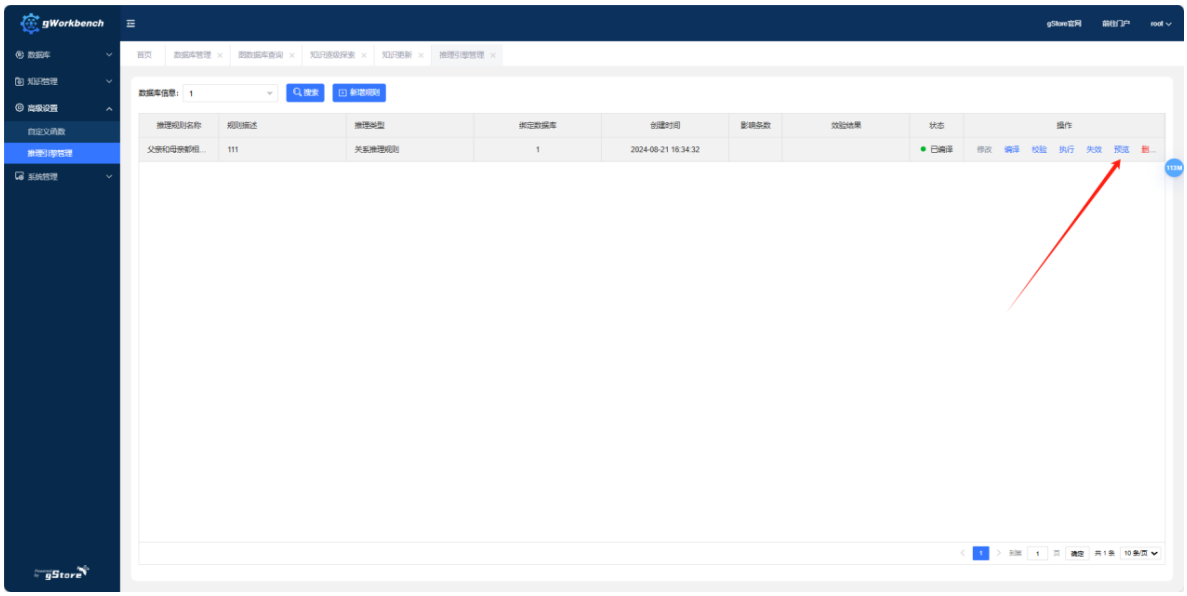
(1) AND (both inference conditions must be met)

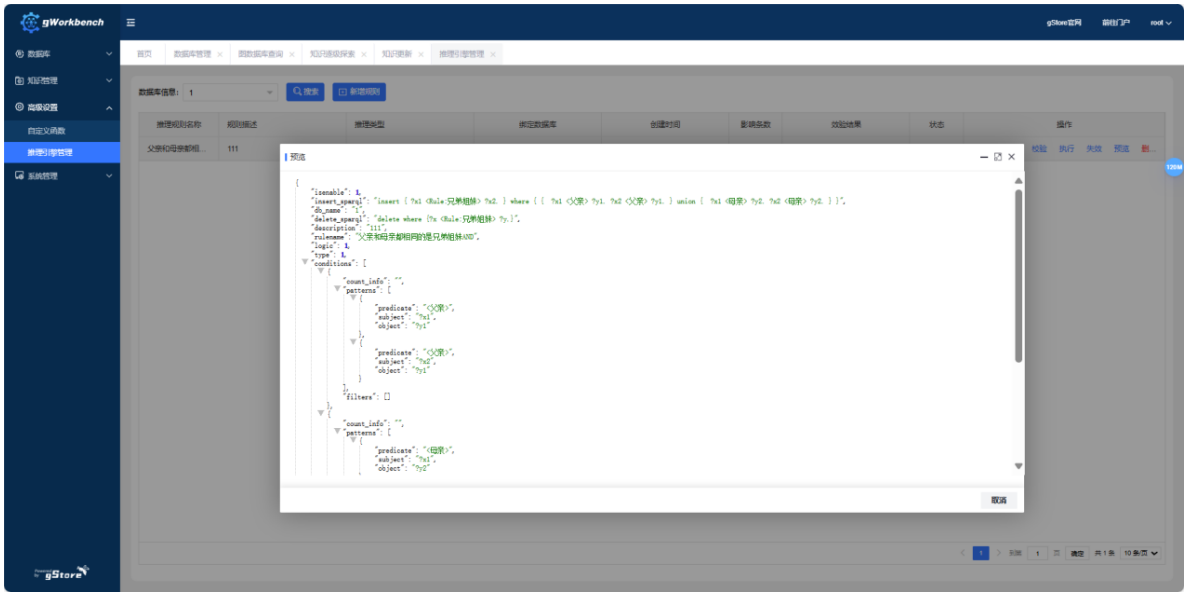


Compile here first, and a statement to be executed will be generated:

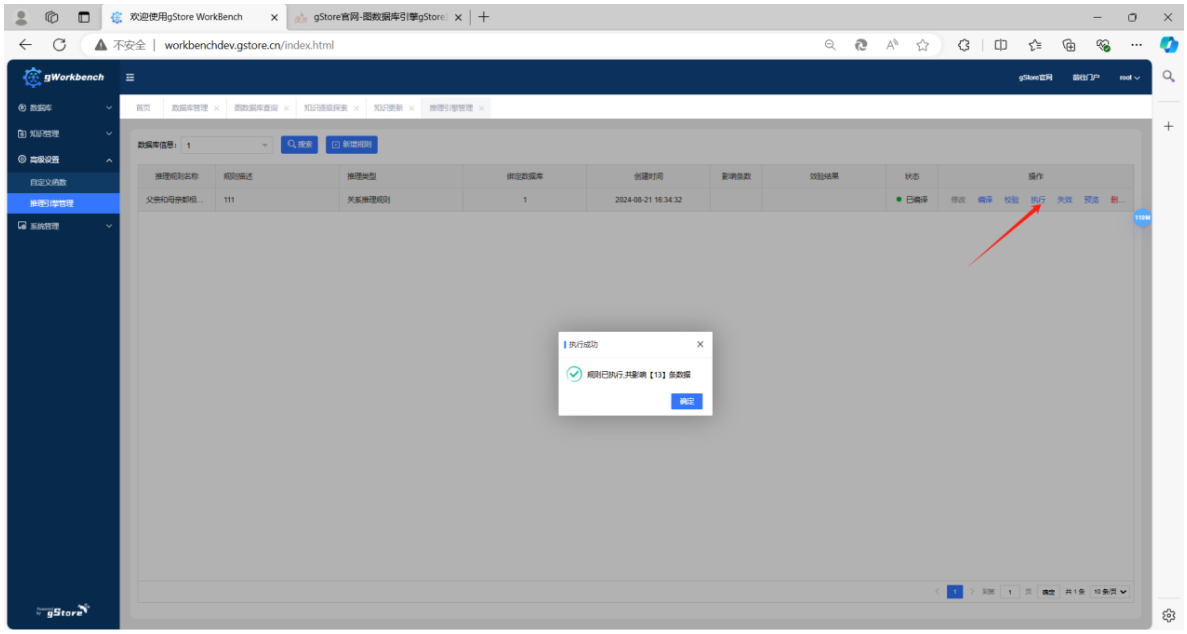


Preview:

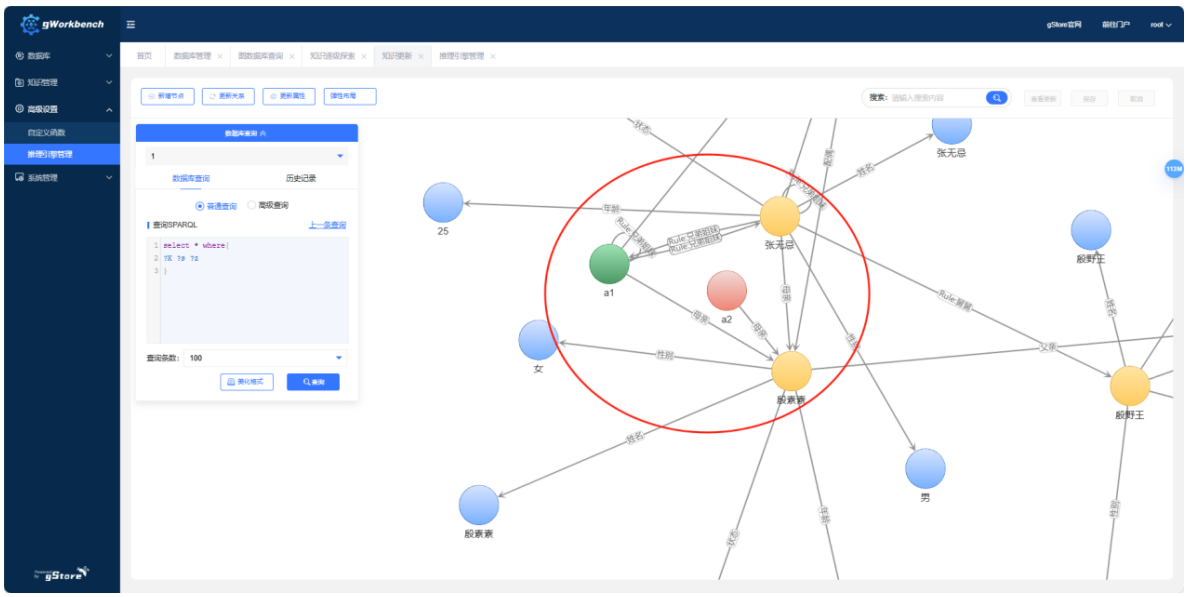




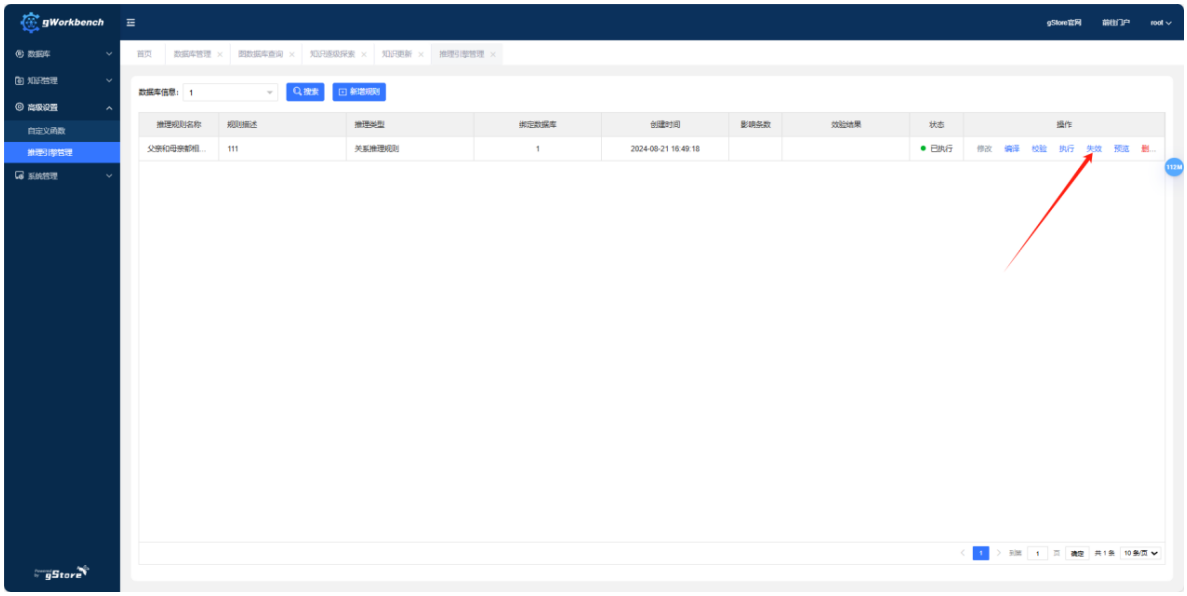
Execute:



The result is as follows:

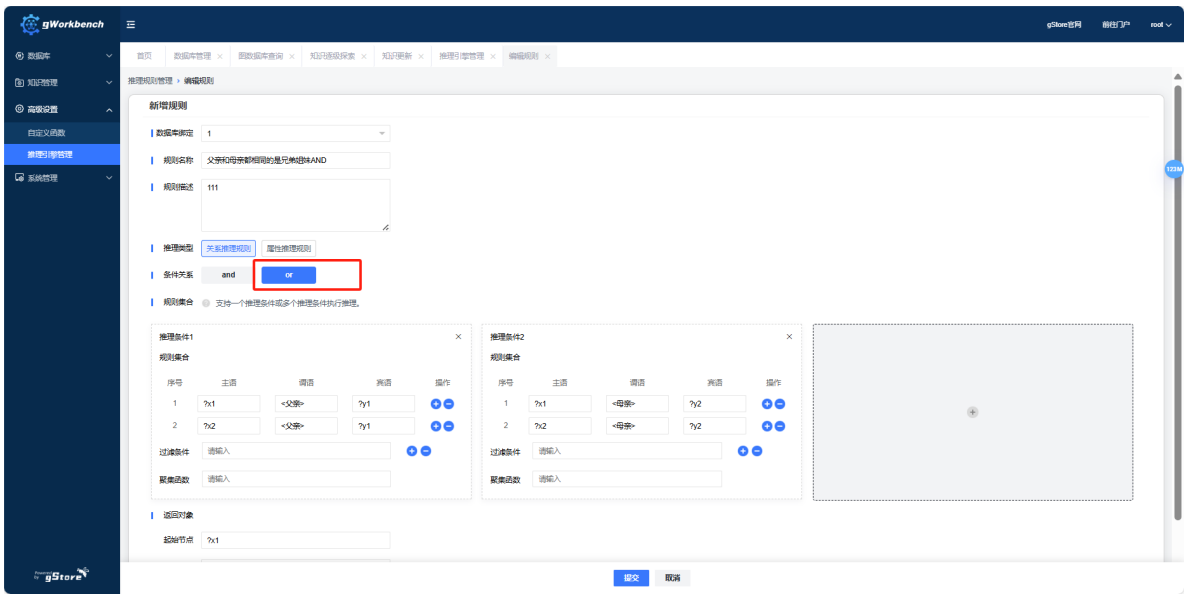


Click "Invalidate" to undo this operation:



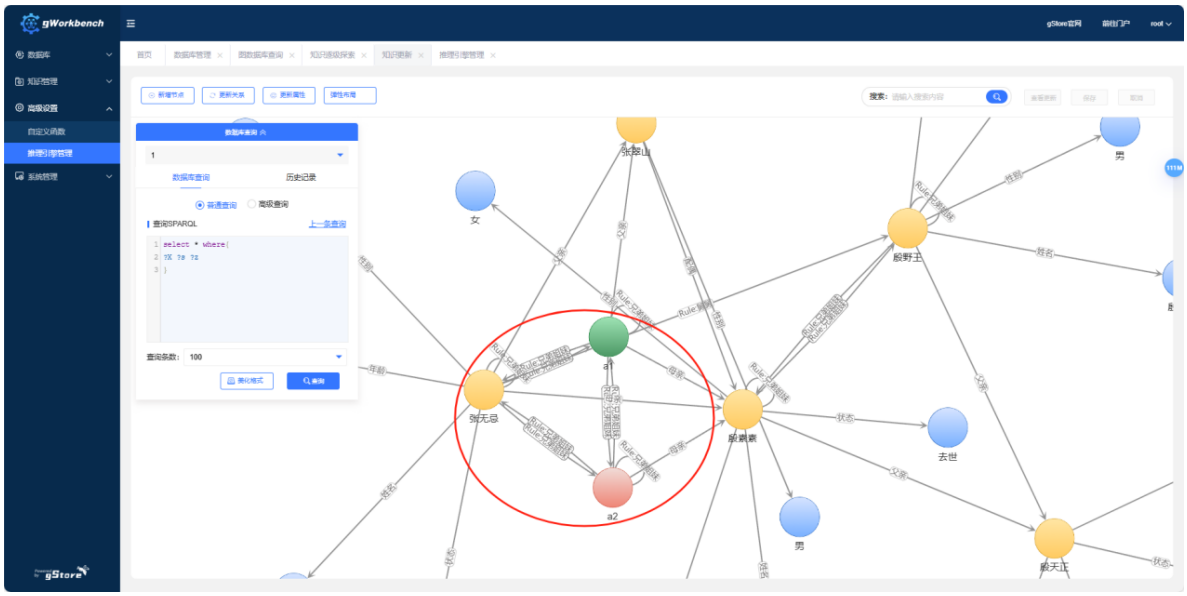
(2) OR (only one of the inference conditions needs to be met)

Simply modify this part of the previous rule:



The remaining steps are the same as for AND.

The result is as follows:

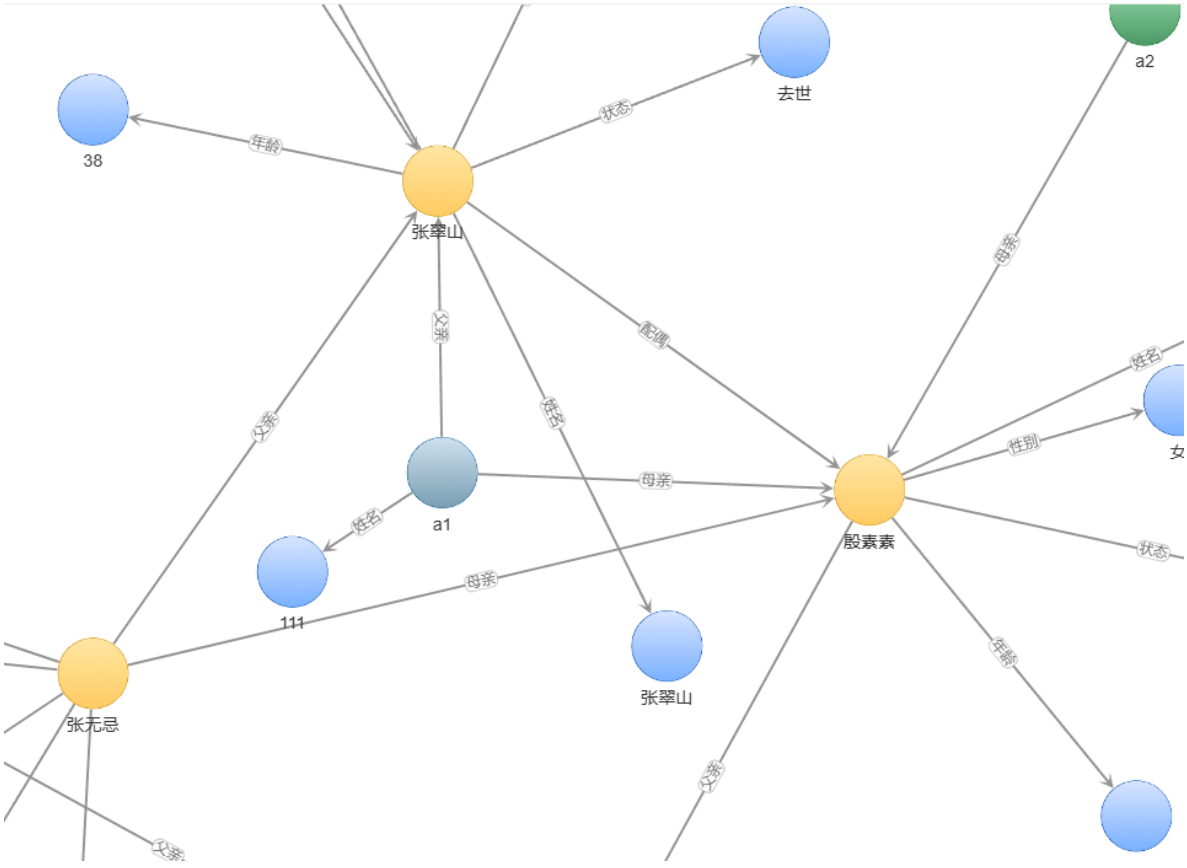


Attribute Rules

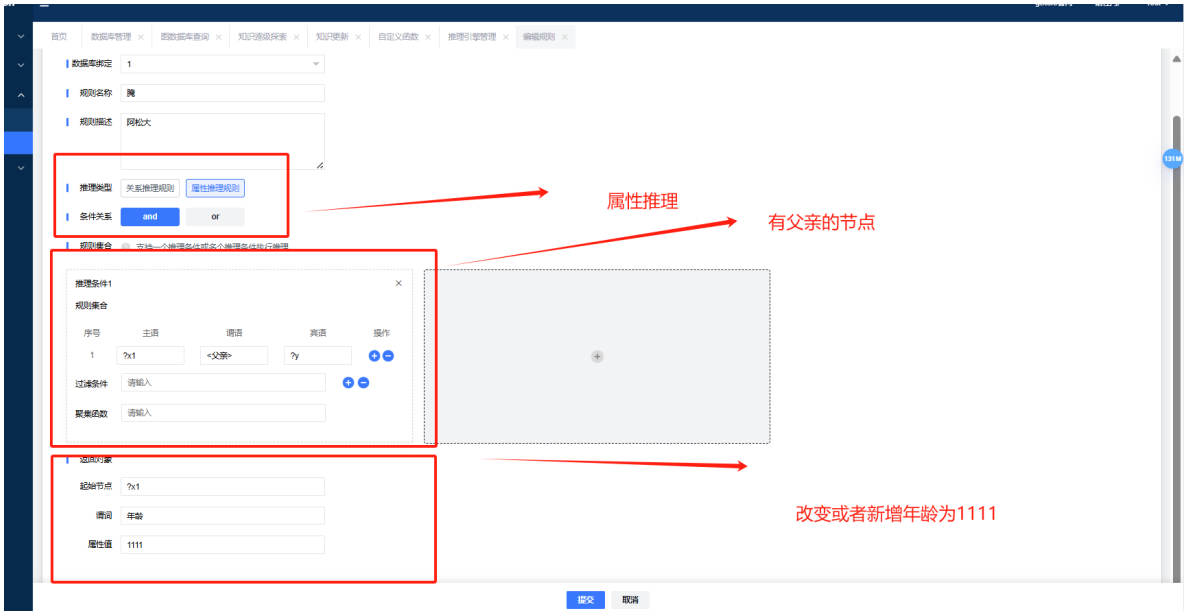
Attribute rules change or add the specified properties of the corresponding node when conditions are met. The other steps are the same as in the previous inference rules, so I will only show the modified pages and results.

Select inference type, input inference conditions, and specify return results.

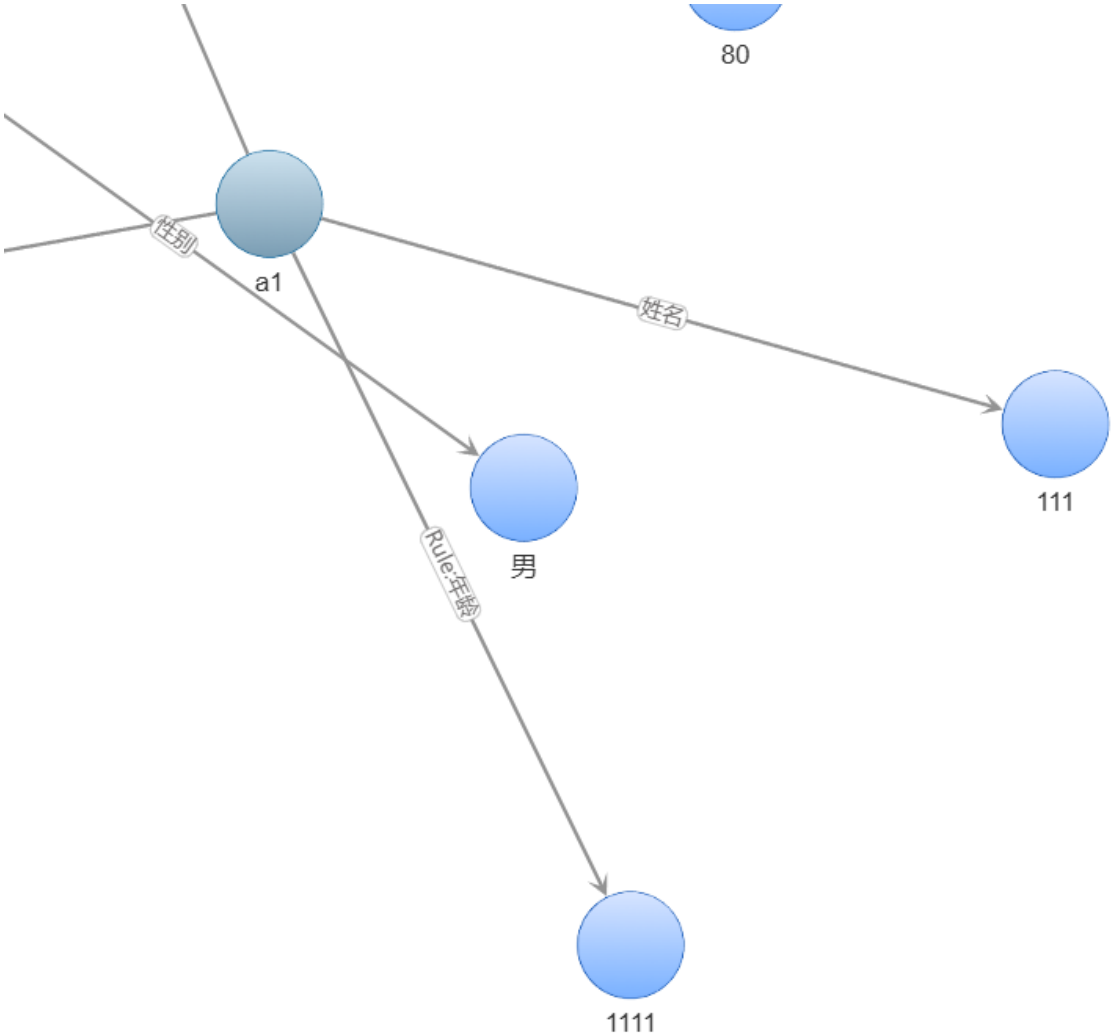
This is the original data:



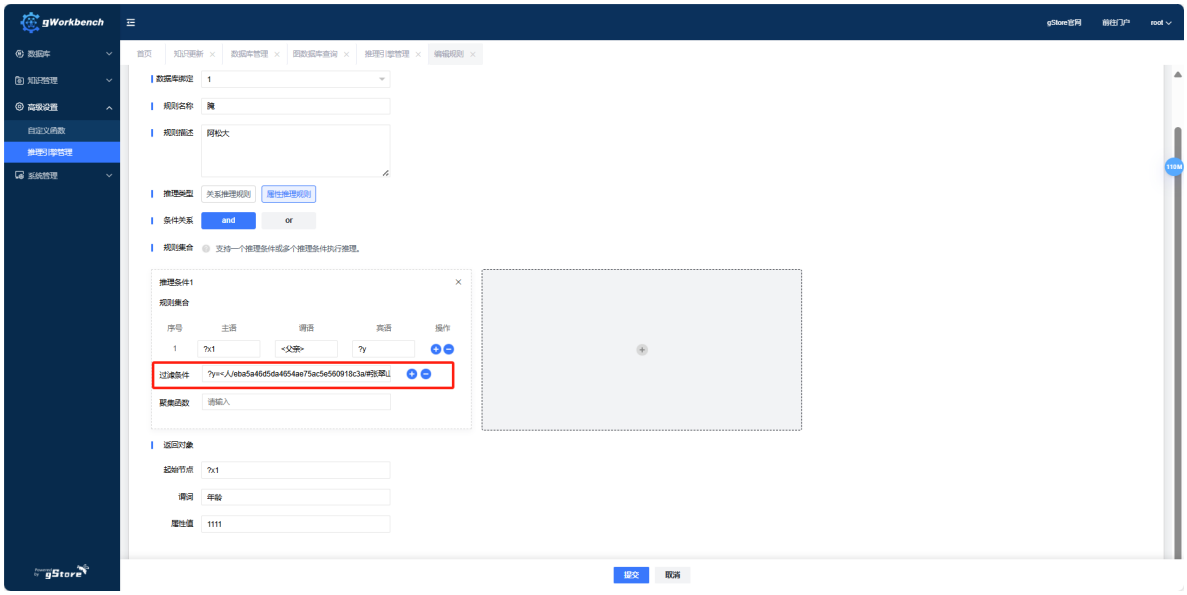
This is the rule:



The result is as follows:



Using a filter condition, for example, where y can only be Zhang Cuishan:



7.5 System management

7.5.1 IP blacklist and whitelist

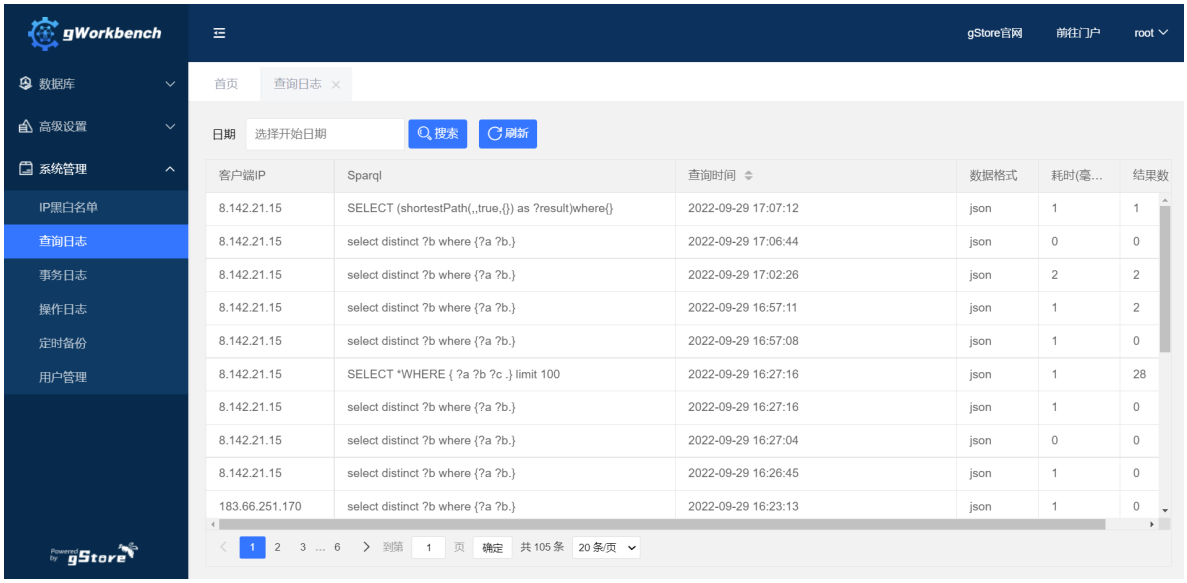
- Users can use the IP blacklist and whitelist function to limit the IP addresses that can initiate accesses. Users can use the blacklist function to prevent blacklisted users from using the system, or you can use the whitelist function to allow accessible IP addresses.

Enter the blacklist and whitelist IP address, separated by commas (,). Range configuration is supported using "-", such as: IP1-1P2.

7.5.2 Log management

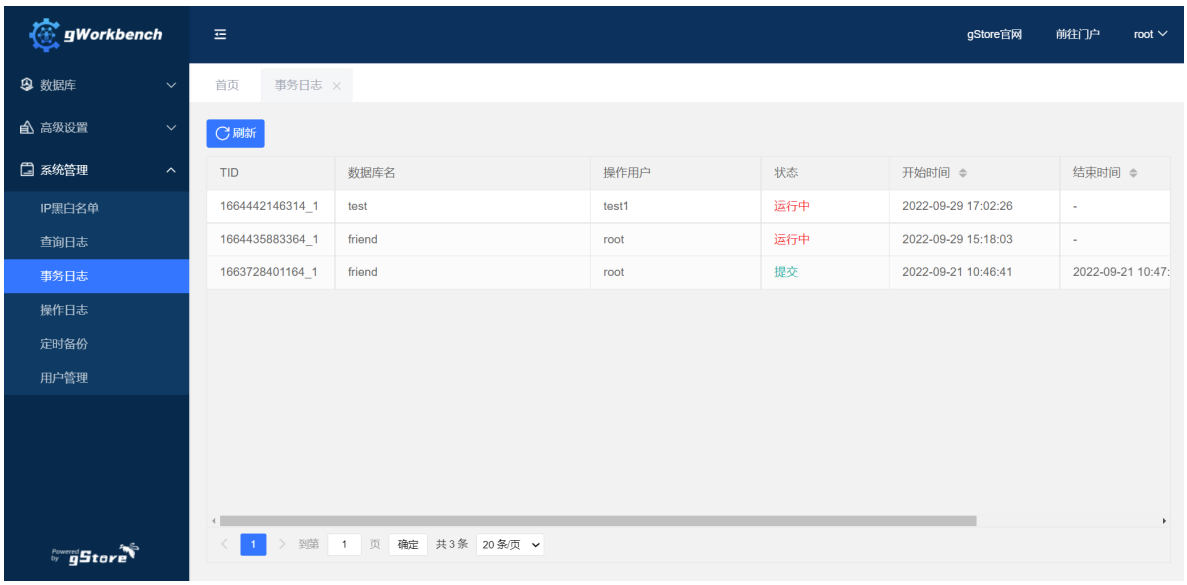
- User can view system logs on the web UI.

Click on the [system management] - [log management], select a specific date in the search bar, and click "Search" to view the specific log information of this date, including name of the log file, the client IP, SPARQL, query time, data format, the execution time (ms), and the number of results.

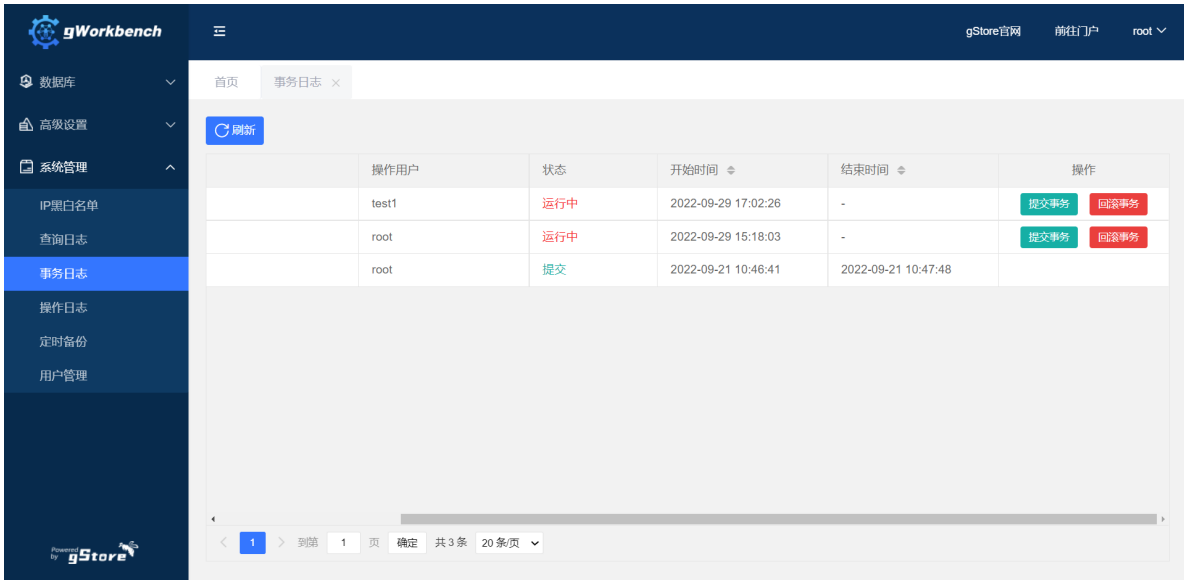


7.5.3 Transaction management

Click [System Management] - [Transaction Management] to view specific transaction information, including TID, database name, the user initiating the operation, transaction status, start time, end time, etc.

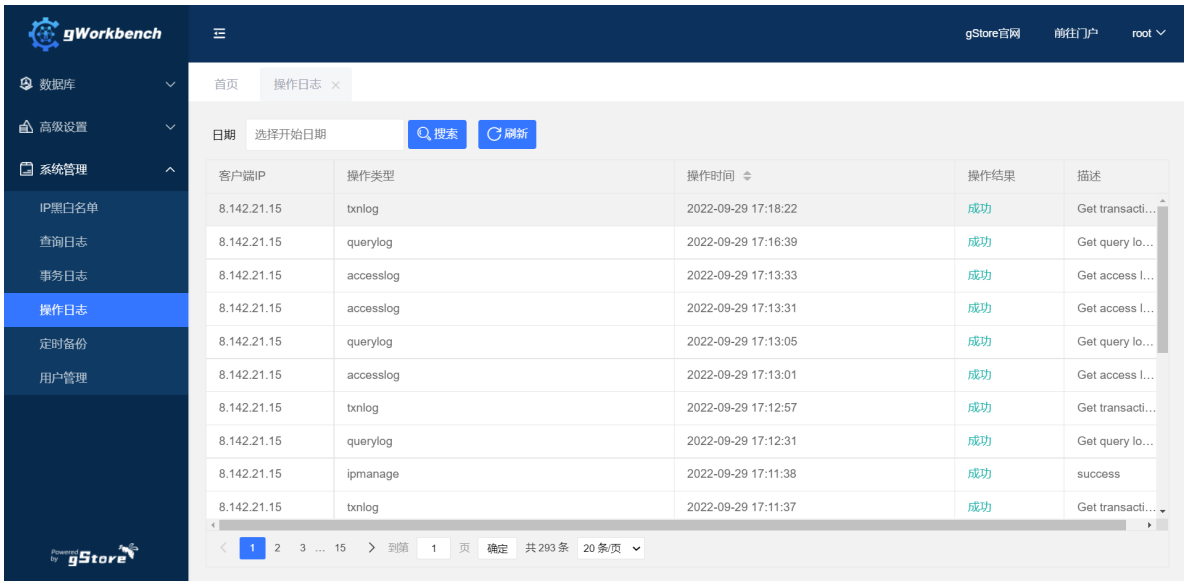


Meanwhile, transactions can also be committed and rolled back.



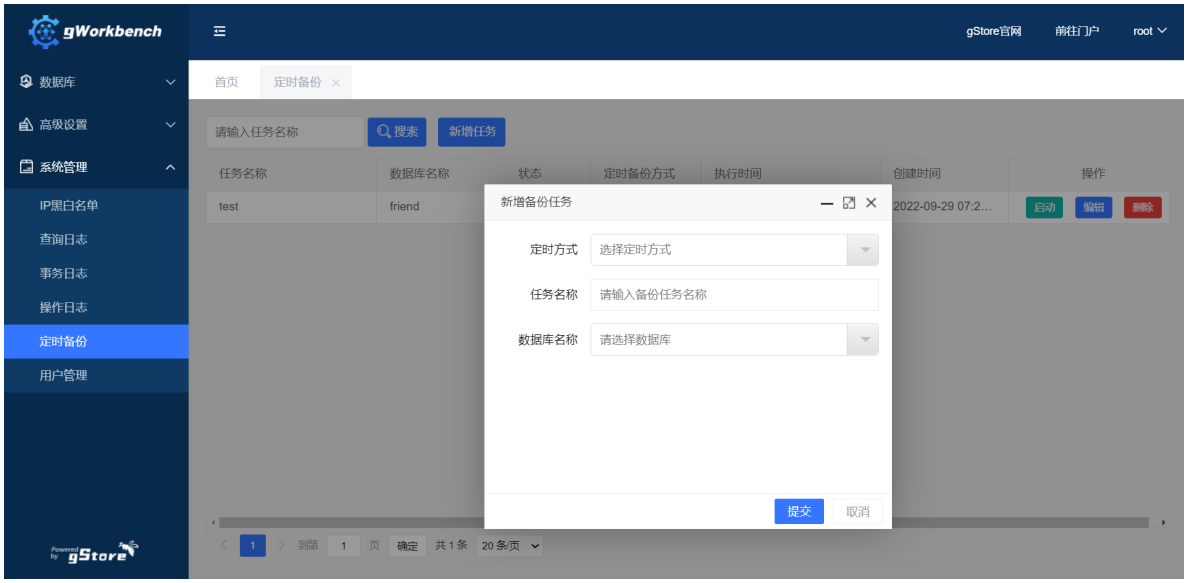
7.5.4 Operation Log

Click [System Management] - [Operation Log], select a specific date in the search bar and click "Search" to view the specific log information of this date, including client IP address, operation type, operation time, operation result and description.



7.5.5 Scheduled backup

Click [Scheduled Backup] and then [New Task] to add a new backup task by filling in the scheduling
mode, the task name, the database name and the backup path.

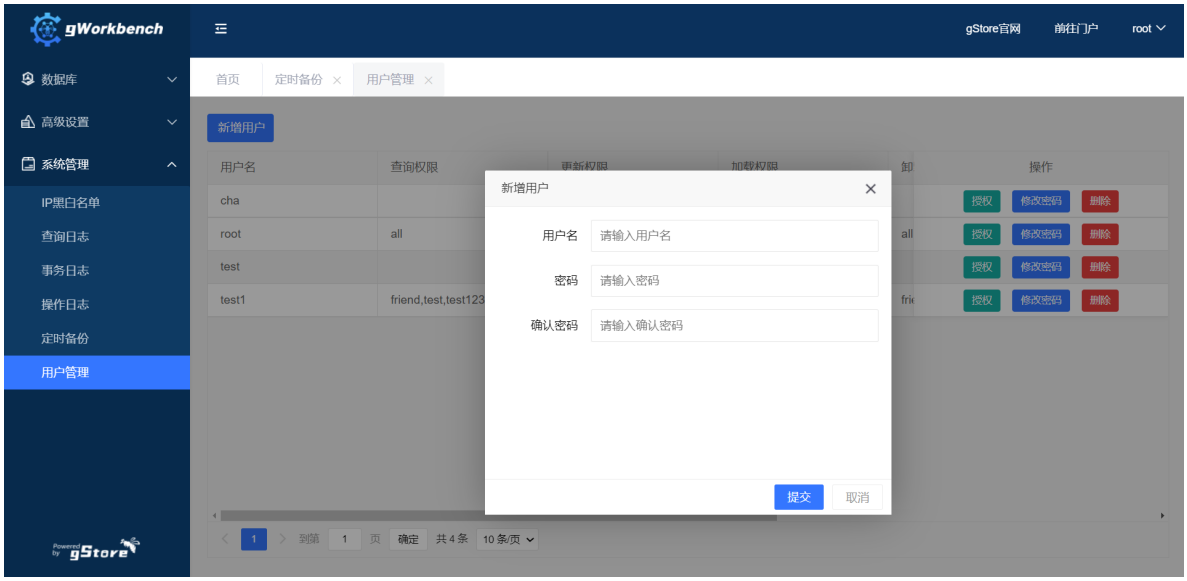


7.5.6 User management (Only for root user)

(1) Add user

- Add new users

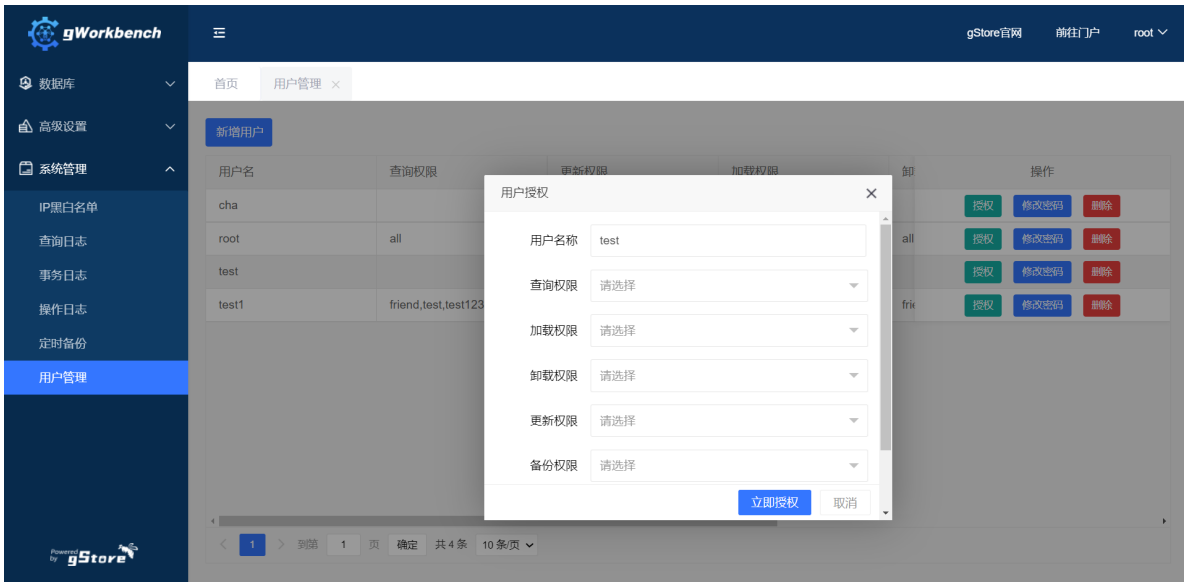
Enter the user name and password to add a user.



(2) User authorization

- Authorize functions for users.

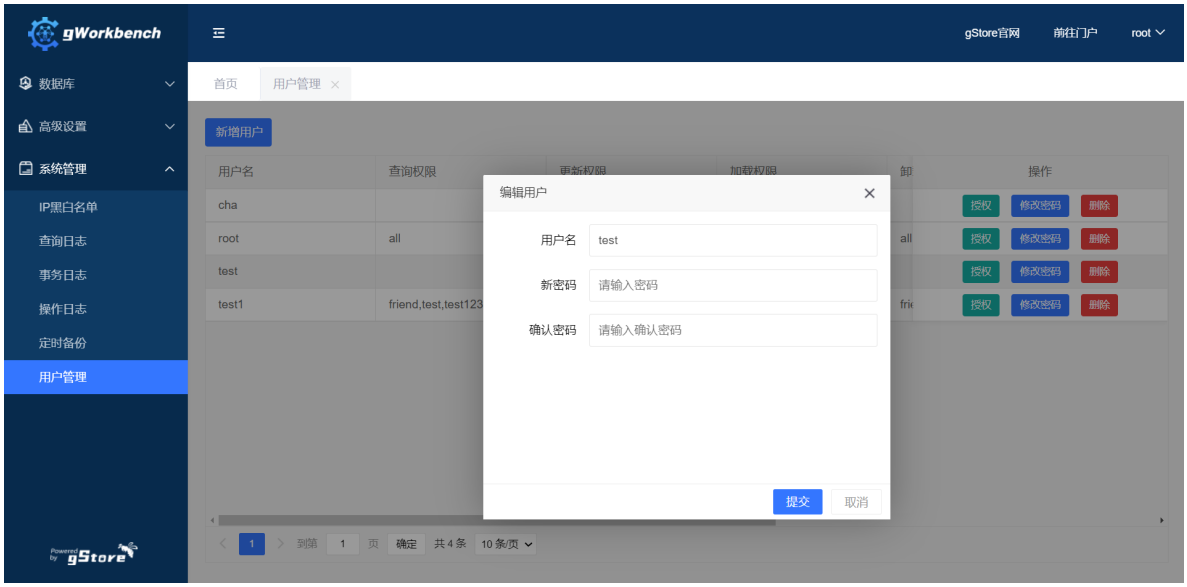
Select the users and databases that you want to authorize, and add or remove query, load, unload, update, backup, restore, or export permissions.



(3) Edit accounts

- Edit user account details.

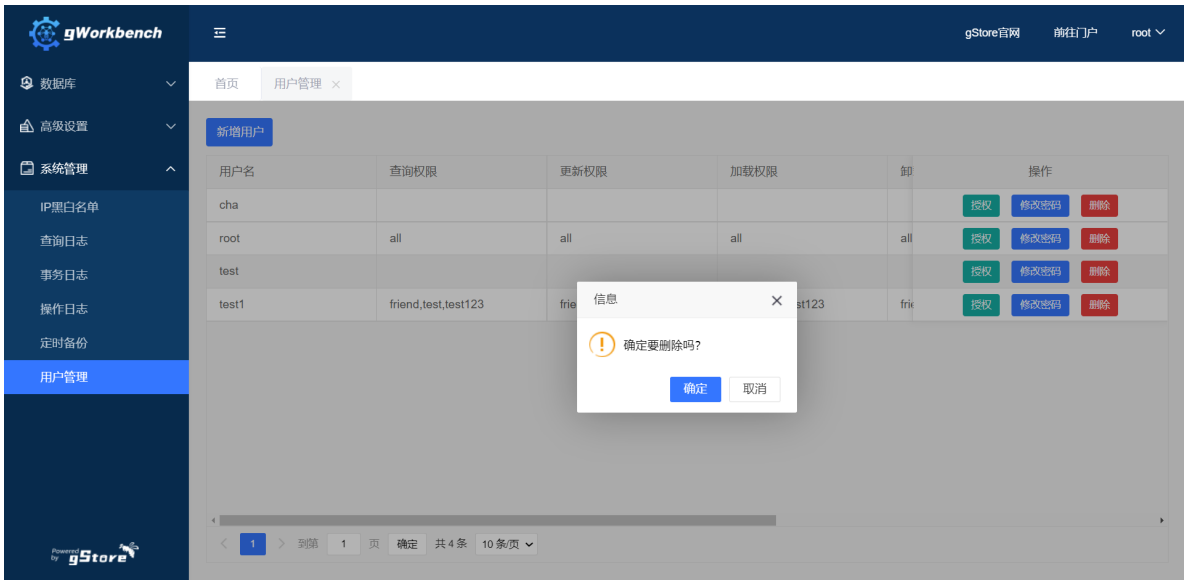
Click [User Management], select a User account, click [Change password] under the operation bar, input relevant information and click [Submit] to modify the user's password.



(4) Delete accounts

- Delete user accounts.

Click [User Management], select a User account, and click [Delete] under the operation bar to delete the User.



8. gStore Cloud platform user manual

8.1 Brief introduction

8.1.1 What is gStore?

gStore is a graph-based RDF triplet storage data management system developed by the Data Management Laboratory of Wangxuan Institute of Computer Technology of Peking University . It can be used to manage huge interconnected data. It has four advantages: original innovation, standard system, superior performance, and independent control

8.1.2 What is gStore cloud platform?

The gStore cloud platform is the cloud service version of the gStore system. It can be used after being registered online and approved. No download or installation is required.

8.1.3 What is the use of gStore

gStore can be used for large-scale data processing, which gives it a wide range of uses, including but not limited to government big data, fintech, smart healthcare, artificial intelligence, etc.

8.1.4 How does gStore play a role in the above transactions?

Taking fintech as an example, the system can query multi-level equity through graph database. In this case, up to five layers of equity relationship data can be found.

8.2 How to use

8.2.1 Registration and Login

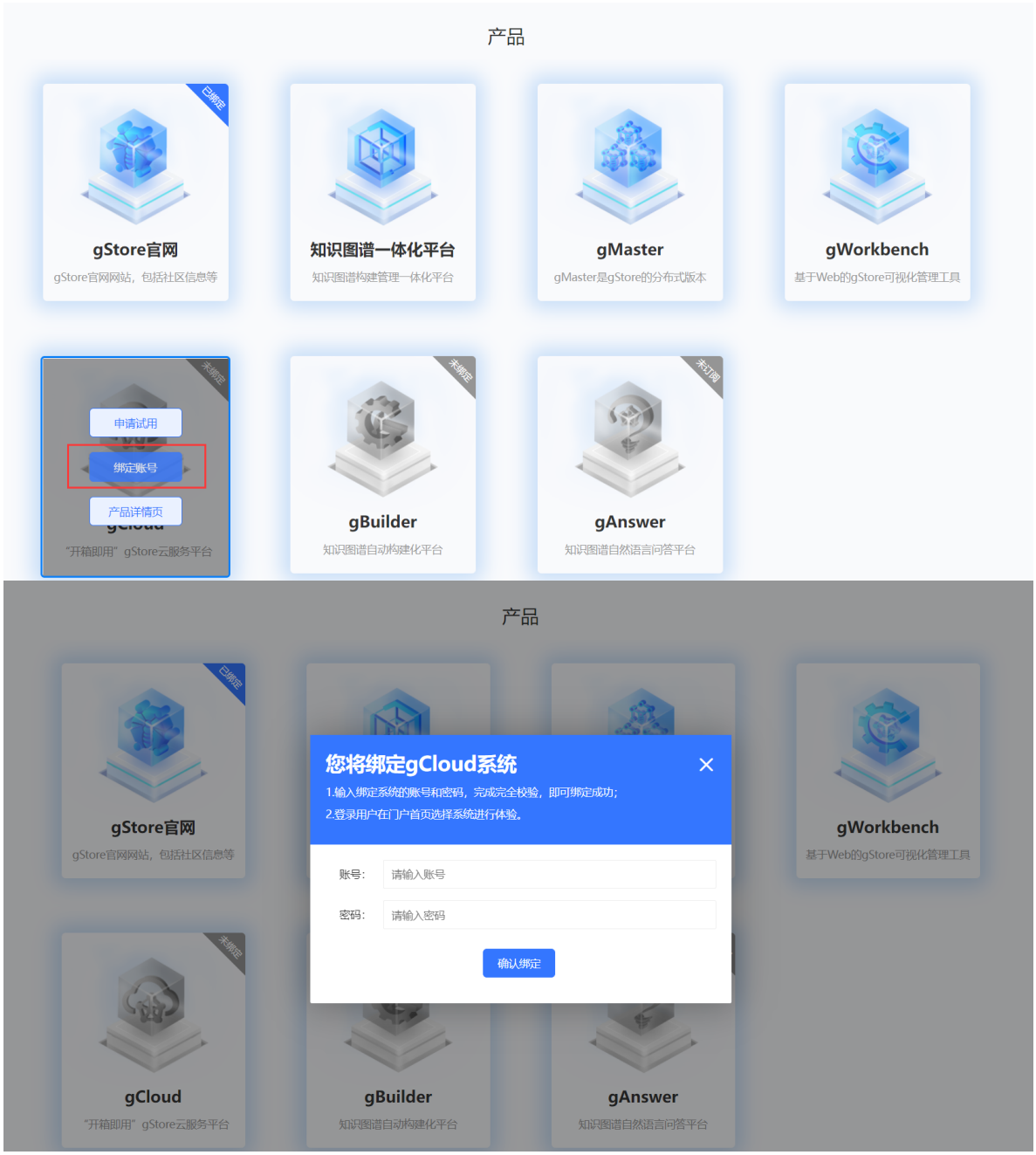
Cloud Platform website：http://cloud.gstore.cn

Users can log in to the cloud platform through the gStore unified identity authentication page.

If you are using the gStore cloud platform for the first time, you need to register. The registration page is as follows:

After registration, you can apply for a trial usage of the cloud platform through the portal:

If you already have a cloud platform account, please bind the account to log in to the cloud platform.



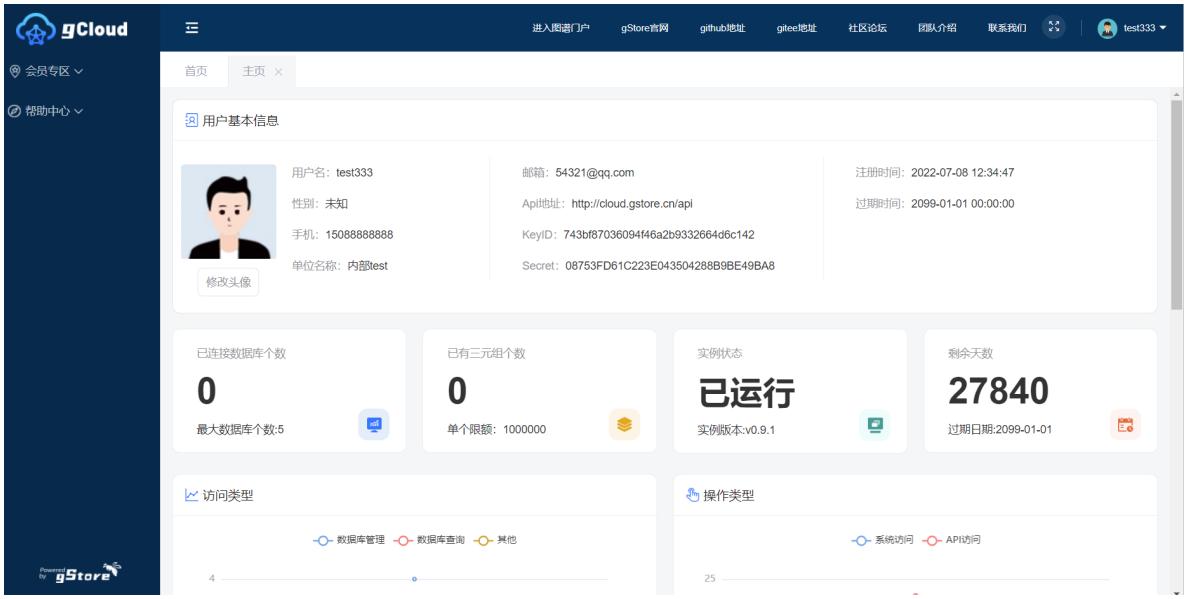
8.2.2 Platform home page

The homepage of the platform is shown in the figure below, which will display the number of currently built databases, the total number of triples and expiration time, as well as information related to the platform (including news and version information, etc.). Click "relevant information" to view the details. In addition, some common links such as the gStore official website, its GitHub and Gitee repository, and community forums can be accessed by clicking on the upper toolbar.

8.2.3 Personal center

The personal center is in the upper right corner of the gStore page.

After logging in to the personal center, you can view the basic user information and operation logs of this week. The basic user information includes KeyID and Secret, which are used as keys when other programs connect with the gStore cloud platform.

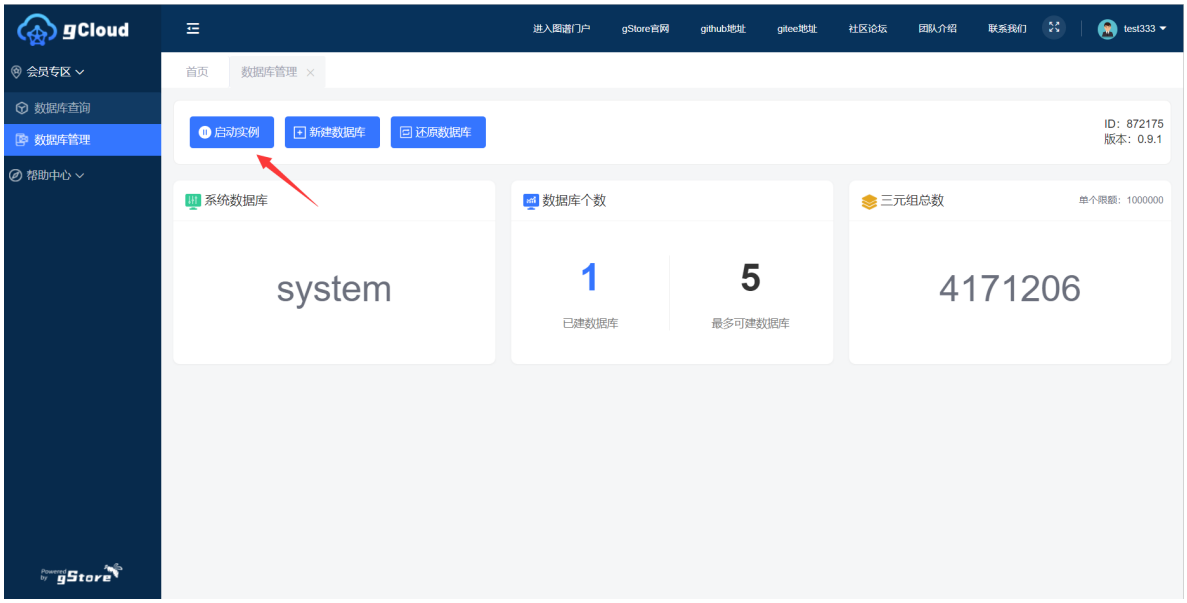


8.2.4 Database management

The area on the left is the system menu, including the members area and help center.
The member area is divided into two functional modules, database management and database query.

(1) Starting a database instance

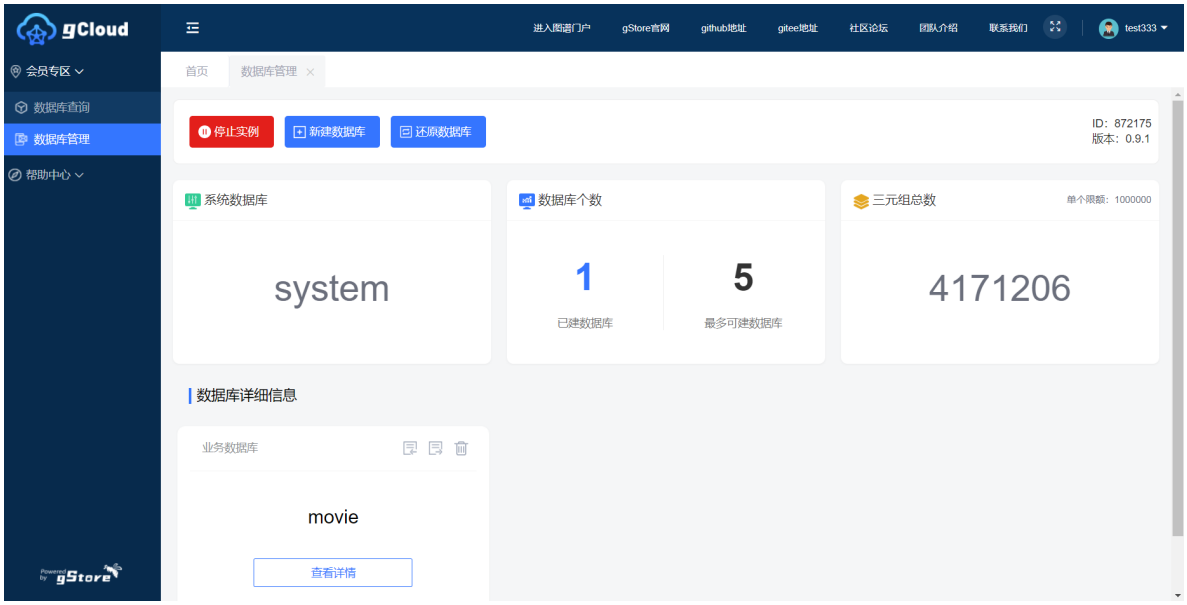
There is an important feature in database management: instances. When first entering the system, the instance may be stopped and need to be started manually.



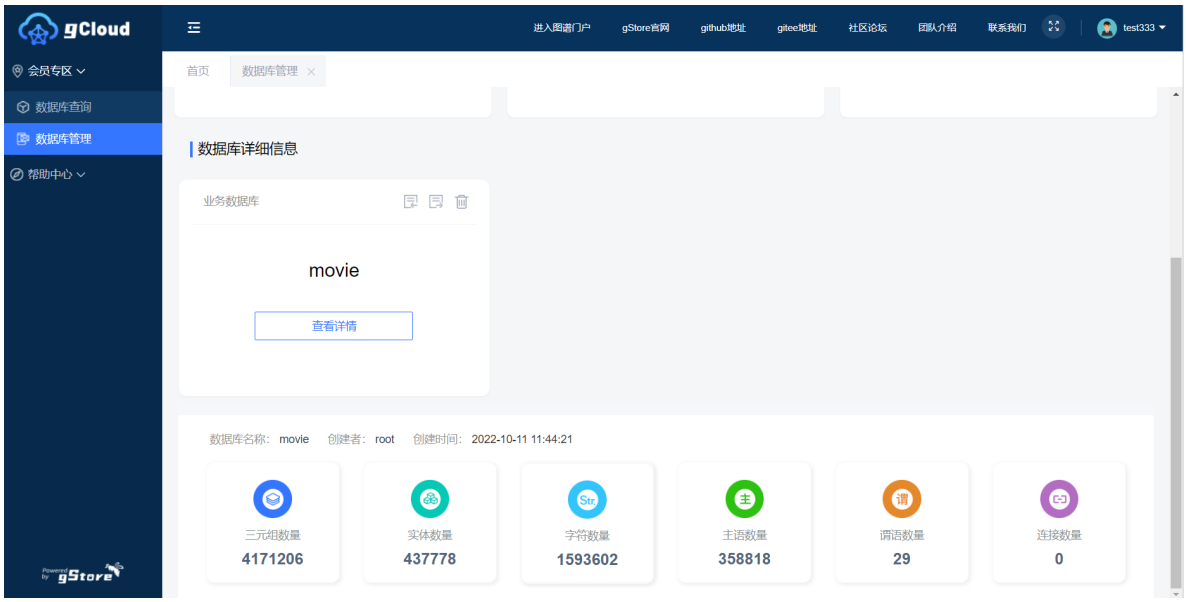
(2) Viewing database Information

After the instance is started, you can see the status of the instance, the number of databases created and the maximum number of databases that can be created, the expiration time of gStore, the total number of triples, and so on in the above row.

Below, you can see the databases that have been created, including a system database (created by the system, not operable, not included in the maximum number of databases that can be created) and several custom databases (created by yourself, operable).

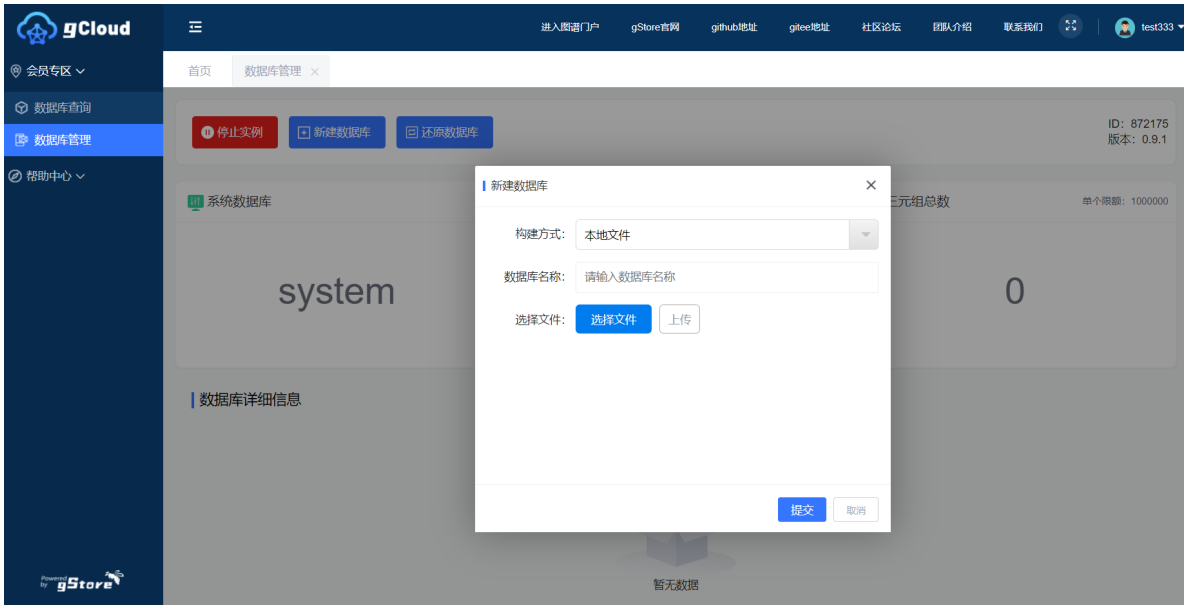


You can create, delete, export, and obtain information about a custom database.
Click on a database to get information about it, including the creator, the creation time, the number of triples, the number of entities, the number of characters, the number of subjects, the number of predicates, the number of connections, and so on. The picture below shows information about the Movie database:

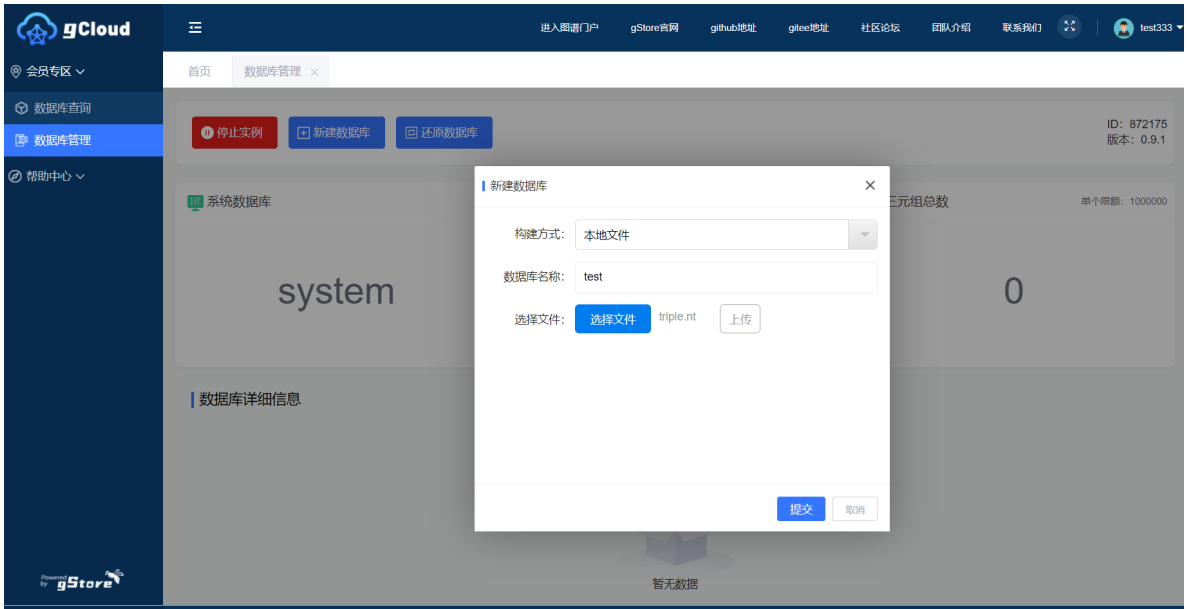


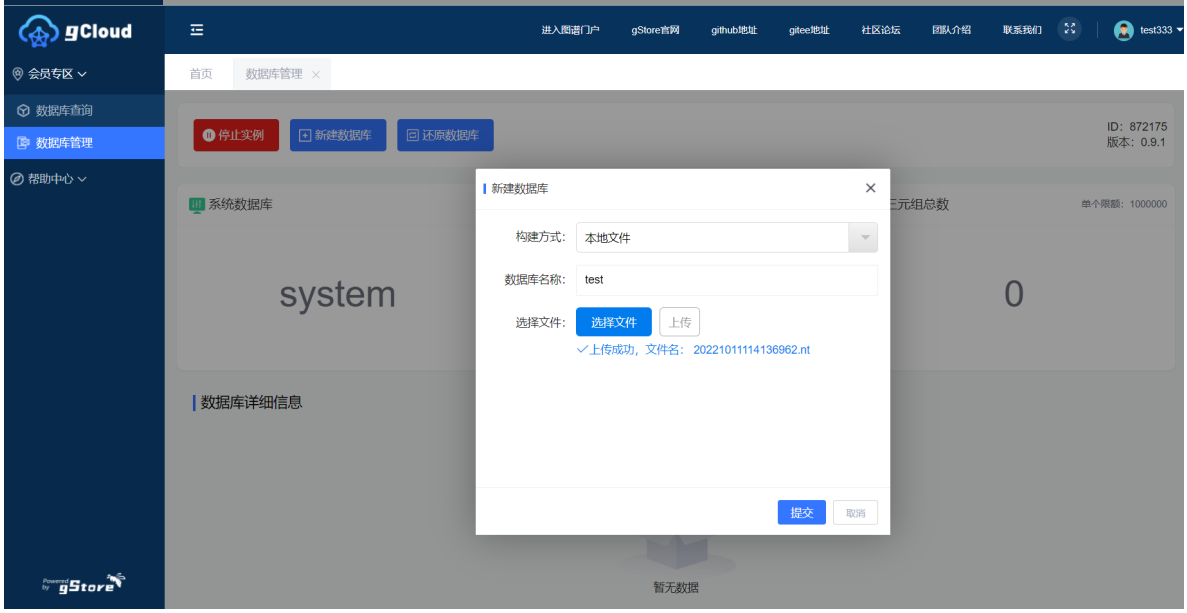
(3) Creating a database

Click [New Database] to create a database: (**Due to limited resources, the number of databases created by each user is limited to 5, and the number of triples of each database is limited to 1 million. If necessary, you can apply for space expansion from the administrator**)

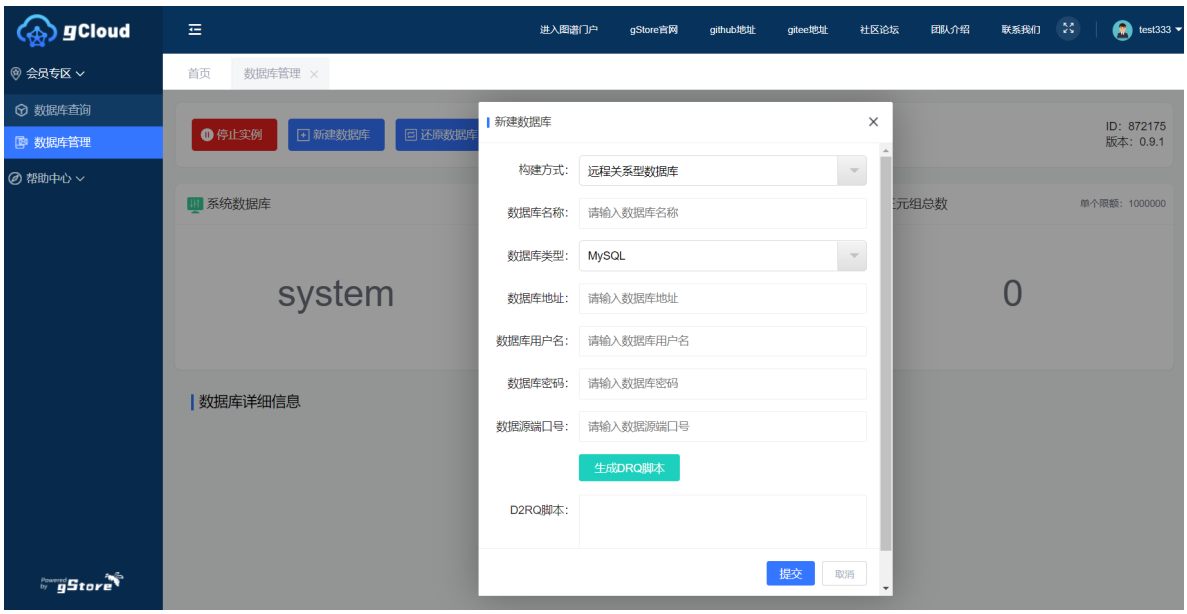


There are three ways to create a database. The first is a local file, that is, upload a file from the local PC to the server. Currently, the system only supports NT files (N3 files may be supported in the future).

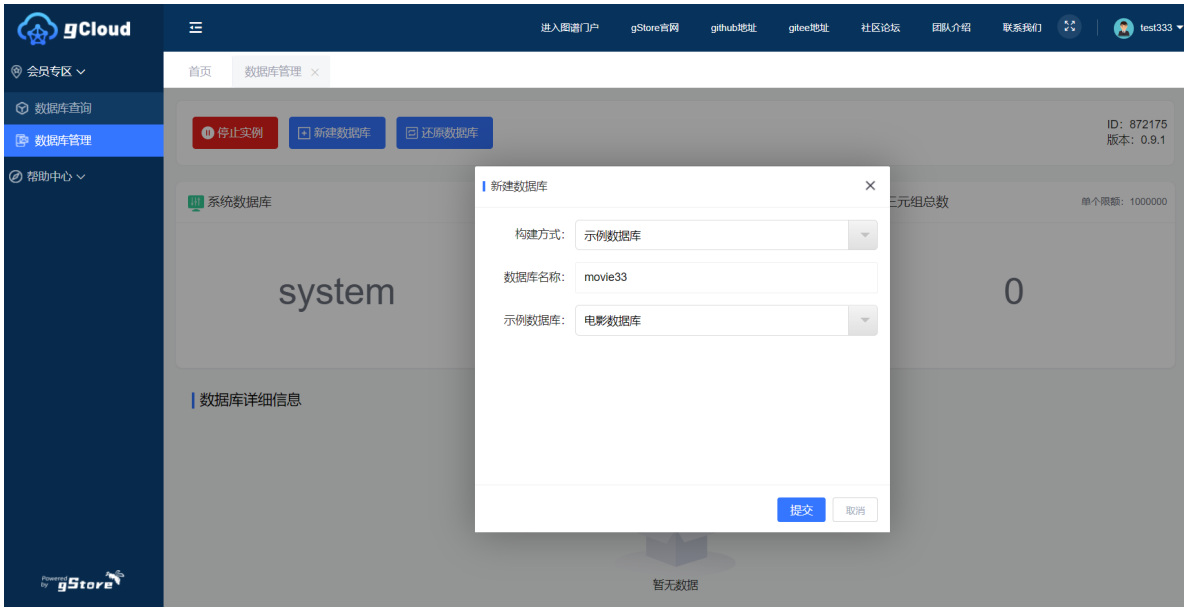




Note that the file size cannot exceed 2GB and the number of lines cannot exceed 1 million.
The second way to create a database is a remote relational database, which remotely accesses a database on the network and imports it into the cloud platform. The cloud platform supports four relational databases: MySQL, Oracle, SQLServer, and Postgre. When creating a database, you need to input its related information and generate a D2RQ script to generate a database.

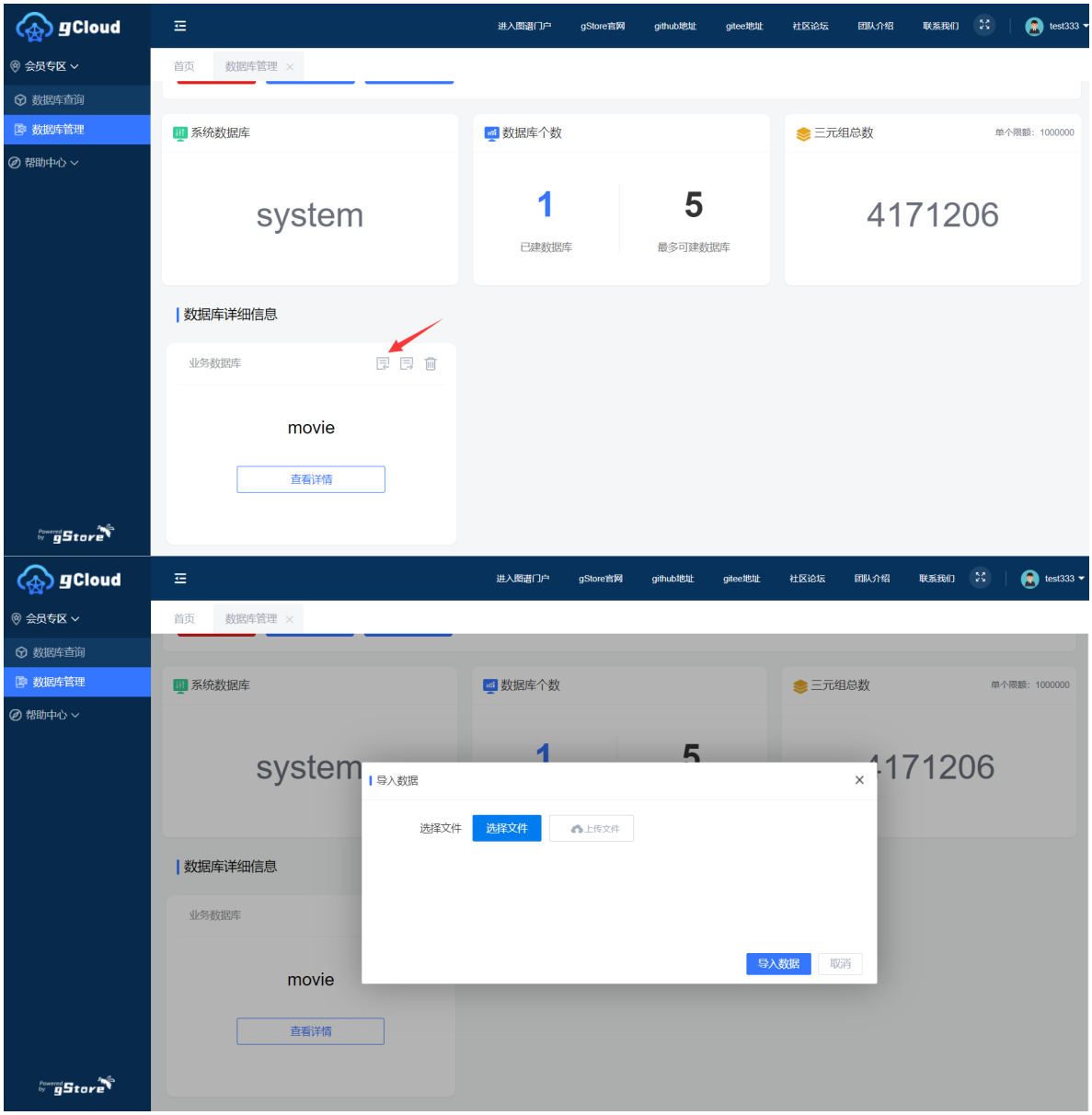


The third way to create a database is to use example database. Currently, the movie database is the only example database in the cloud platform, but more will be added over time. The movie database contains more than 4 million triples containing information about movies, directors, actors, release dates, movie ratings, and more. (**Sample database triples are not limited to 1 million triples per database**)



(4) Importing a database

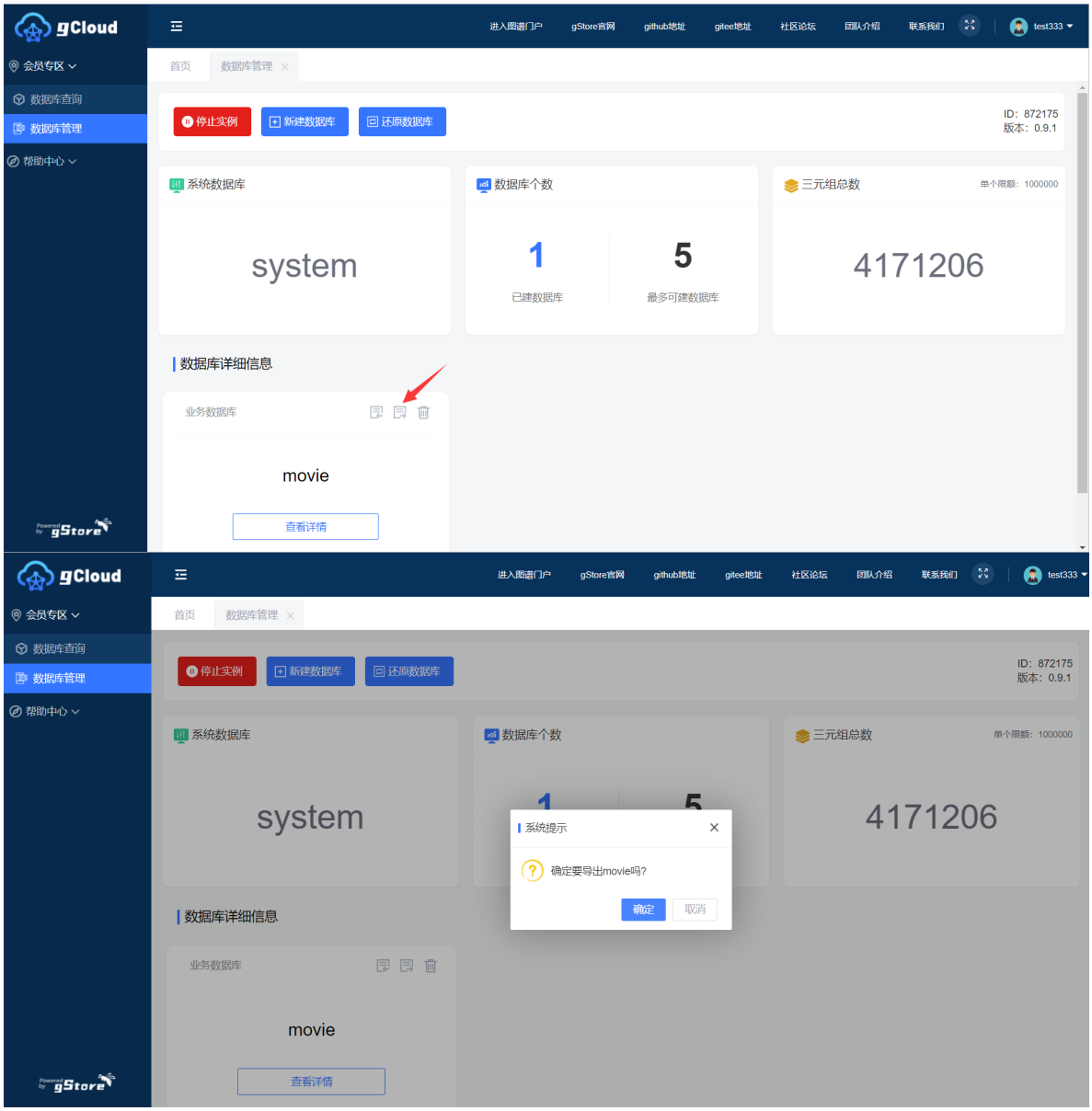
You can import data to an existing database by clicking the up arrow icon in the upper right corner of the database.



Select file upload and click [Import Data].

(5) Exporting a database

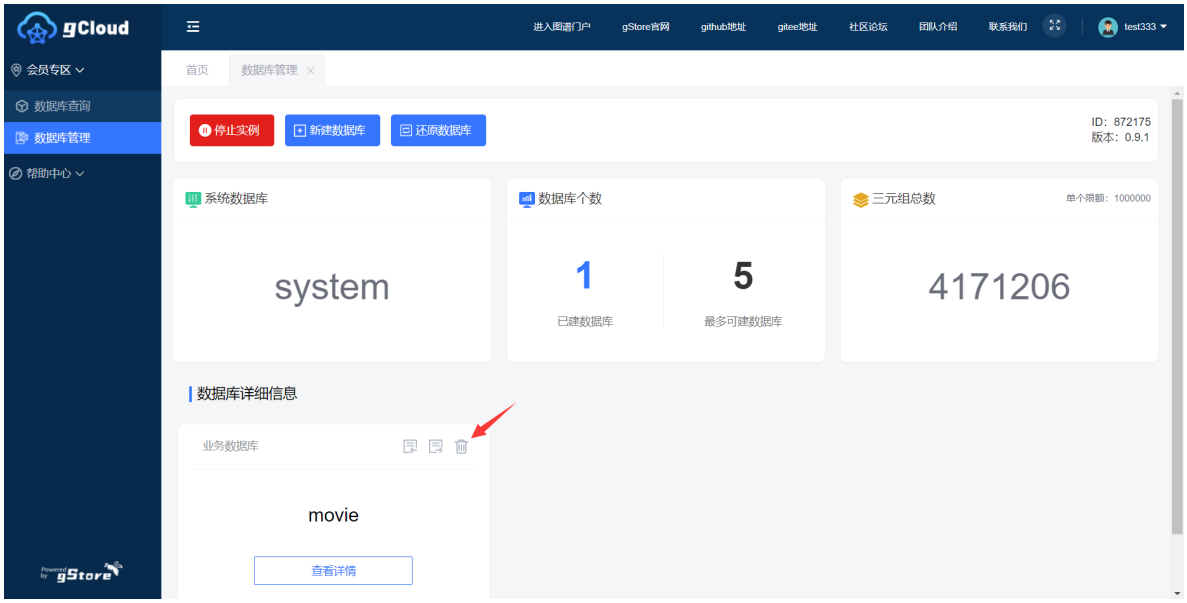
You can export a database by clicking the down arrow icon in the upper right corner of the database.



After clicking Export, a zip file will be created, downloaded and decompressed to get the NT file corresponding to the database. After that, more database functions such as database rename and database backup will also come online.

(6) Deleting a database

You can delete a database by clicking on the trash can icon in the upper right left corner of the database.
The system provides a 15-day recovery period for all deleted databases to prevent accidental deletion.



8.2.5 Database query

In terms of database query, gStore cloud system provides a visual query interface. Enter SPARQL statements in the text box below to get results. (Note: Considering system performance, the diagram and JSON data only show 100 result entries; you can click the [Download] button next to the JSON to get all the returned data)

8.3 End

This is the end of the help manual for gStore Cloud platform. If you have any questions about the use of the cloud platform, you can click the "Submit questions" in the upper right corner of the cloud platform to give your opinion in the community.

9. gStore Chronology

Year 2024

- September, gStore 1.3 version released

Year 2023

- November, gStore 1.2 version released
- November, gStore selected as an outstanding achievement in the annual world open source system by BenchCouncil
- May, gStore held a product launch event with the theme of "linking data to create value" for domestic high-performance graph database system

Year 2022

- In October, gStore 1.0 version released
- In August, the knowledge graph integrated platform released

Year 2021

- In November，gStore 0.9.1 version released
- In October，gBuilder 2.0 version released
- In February，gStore products have completed the "Graph Database Basic Ability Test" project of China Academy of Information and Communication Technology；
- In February，gStore's new official website launched；

Year 2020

- - In December, gStore added advanced query functions such as shortest/longest path, K-hop reachable query and loop detection to further enrich the gStore algorithm library;

  - In December, gStore Beta (V0.9) and gStore Stable (V0.8) were officially released on Github and Gitee;

  - In November, gBuilder V0.1 version of knowledge graph automation construction platform was launched;

  - In October, gStore distributed version gMaster was demonstrated in related projects of institute of Computing Technology, Chinese Academy of Sciences;

  - In July, gStore was successfully adapted with TONGxin UOS operating system and domestic CPUS of Kunpeng/Haiguang/Zhaoxin/Feiteng

Year 2019

- - In December, PKU graphics database system gStore launched China Science and Technology Cloud 2.0;

  - In November, China Software Evaluation Center conducted a performance test on gStore distributed system, and the test results showed that the average query response time of gStore distributed system was 1.79 seconds under the condition of 10.6 billion data storage scale;

  - In October, pKU Graphics database system gStore cloud platform was deployed and launched;

  - In September, THE GRAPH database system was successfully adapted with the domestic "PK "system (Feiteng CPU+ Kirin operating system);

Year 2018

- Multi-query Optimization in Federated RDF Systems 23rd International Conferenceon Database Systems for Advanced Applications (DASFAA) BEST PAPER AWARD

  - Related theoretical research work of gStore system "Large-scale Graph structure Data Management", won the second prize of Natural Science of Ministry of Education of China (Zou Lei ranked first)

  GAnswer system is officially open source on Github with version NUMBER V0.1

  - gAnswer system took part in the Knowledge base Natural language question contest QALD-9 held by eu and won the first prize

Year 2017

- The PKUMOD research team has released gStore milestone V0.5 on Github.

Year 2016

- PKUMOD research team was funded by the Key research and development project "Key Technology and System of Graph Data Management" of Ministry of Science and Technology of China.

Year 2015

- gStore code is officially open source on Github, version 0.1

Year 2014

- - PKUMOD Graph data management related theoretical research "Massive graph structure data storage and query optimization theory research", won the second prize of Natural Science of China Computer Society (Zou Lei ranked first)

  - The first academic paper related to natural language question answering based on knowledge graph was published

  Lei Zou, Ruizhe Huang, Haixun Wang, Jeffery Xu Yu, Wenqiang He, Dongyan Zhao, Natural Language Question Answering over RDF ---- A Graph Data Driven Approach, SIGMOD 2014

  - PKUMOD research team was supported by the National Natural Science Foundation of China (NSFC) project "Key Technology research on Graph Based Matching Query for Large-scale Heterogeneous Information Network"

Year 2011

- - The first academic paper of gStore was published

  Lei Zou., et al., gStore: Answering SPARQL Queries via Subgraph Matching. PVLDB 4(8): 482-493 (2011)

  - PKUMOD research team was supported by the "Research on Massive RDF Data Storage and query Method based on Graph Database Theory" project of natural Science Foundation of China (NSFC)

10. Open source and legal provision

10.1 Open Source and Community

gStore system from January 2015 open source in the making, comply with the BSD, 3 - Clause open source licenses https://github.com/pkumod/gStore is open source address; We advocate users to use and modify gStore freely on the premise of respecting the copyright of code authors, develop various industry applications of knowledge graph based on gStore, and promote the healthy and sustainable development of knowledge graph industry software. We encourage users to actively use the gStore system, report problems, make suggestions, and contribute code to the gStore open Source project to join us and make the gStore system better.

If you have any questions in the process of using gStore, if you would like to tell us your name, organization, purpose of using gStore and email address, we will timely reply to you by sending an email to service@gstore.cn. We guarantee that the privacy of you and your company will not be disclosed, only used to enhance the gStore system itself

10.2 Legal Issues

The gStore system always adopts the BSD 3-clause which is widely used in the open source community. Under this Agreement, User is free to modify and redistribute the Code, subject to the following terms, and user is free to develop, distribute and sell commercial software based on the gStore code. The specific terms are as follows:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the Peking University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

中文条款声明如下（具有同等法律效应）：

11. gStore Logo

11.1 gStore's Logo

11.2 Powered by gStore Recommend logo